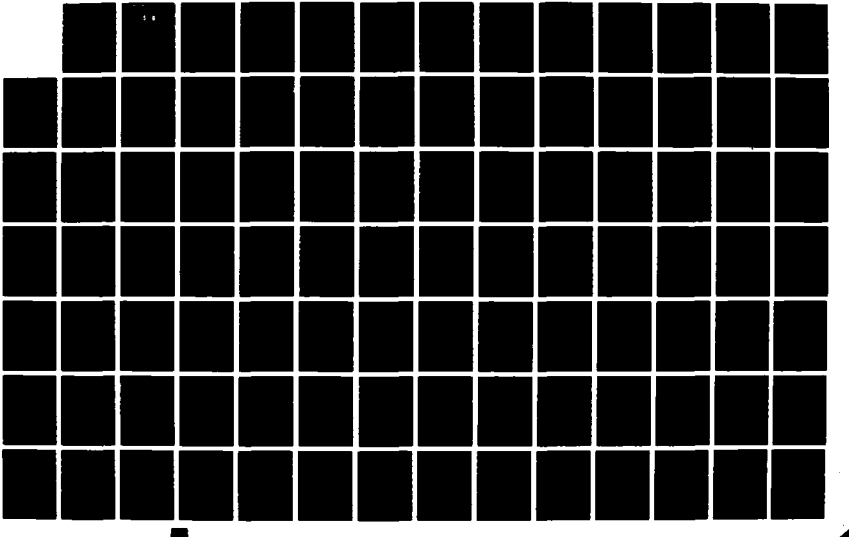
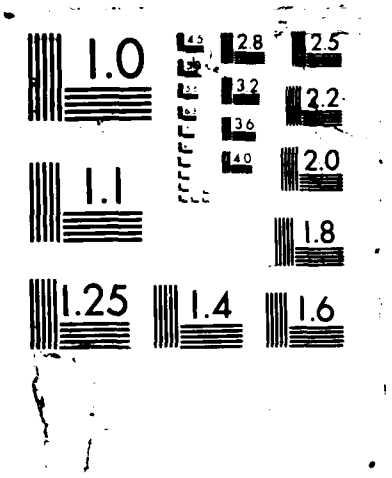
AD-A191 320 PIECEWISE LINEAR APPROACH FOR TIMING SIMULATION OF VLSI (VERY-LARGE-SCALE..(U) ILLINOIS UNIV AT URBANA COORDINATED SCIENCE LAB O TEJAYADI DEC 87 1/2

UNCLASSIFIED UILU-ENG-87-2279 N00014-84-C-0149 F/G 20/6 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

December 1987

UILU-ENG-87-2279
DAC-8

(2)

COORDINATED SCIENCE LABORATORY
College of Engineering

PIECEWISE LINEAR APPROACH FOR TIMING SIMULATION OF VLSI CIRCUITS ON SERIAL AND PARALLEL COMPUTERS

Ongky Tejayadi

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-87-2279 (DAC-8)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b OFFICE SYMBOL (*If applicable*) N/A	7a. NAME OF MONITORING ORGANIZATION Semiconductor Research Corporation Joint Services Electronics Program
6c. ADDRESS (*City, State, and ZIP Code*) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (*City, State, and ZIP Code*) SRC: Research Triangle Park, NC 27709 JSEP: Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION SRC and JSEP 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER SRC: 86-12-109 JSEP: N00014-84-C-0149
8c. ADDRESS (*City, State, and ZIP Code*) See block 7b.	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11 TITLE (*Include Security Classification*)
PIECEWISE LINEAR APPROACH FOR TIMING SIMULATION OF VLSI CIRCUITS ON SERIAL AND PARALLEL COMPUTERS

12. PERSONAL AUTHOR(S)
Tejayadi, Ongky

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Technical	FROM _____ TO _____	December 1987	138

16. SUPPLEMENTARY NOTATION

17 COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	timing simulation, piecewise linear techniques, dynamic partitioning, parallel algorithms, VLSI

19 ABSTRACT (*Continue on reverse if necessary and identify by block number*)

The work presented in this report deals with the development of a fast and fairly accurate Computer-Aided Design software for simulating very-large-scale-integrated (VLSI) circuits. The methods rely on piecewise linearized nonlinear elements in the circuits. The piecewise linear approaches explored in this work are: (1) a fast piecewise linear Gauss-Seidel waveform relaxation method; (2) a slower but more accurate piecewise linear method based on simplices; and (3) a Gauss-Seidel piecewise linear method with dynamic partitioning. Also described is a mixed method which combines the fast piecewise linear method and the dynamic partitioning method. The circuit to be analyzed is partitioned into dc-connected subcircuits and then sequenced for analysis. Small subcircuits are solved using the fast piecewise linear method while large subcircuits, including the strongly connected components in the circuit, are solved using the dynamic partitioning method. A parallel implementation of the Gauss-Seidel piecewise linear method with dynamic partitioning on

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (*Include Area Code*) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

a uniprocessor computer is studied. Algorithms for the parallel implementation of the dynamic partitioning approach on a multiprocessor with shared memory (Alliant FX/8) are also explained in detail. The piecewise linear methods presented in this work have been implemented in a set of programs called PLATINUM. The waveforms generated by PLATINUM are fairly accurate as compared to those for SPICE2, and the speedup for a uniprocessor machine is over two orders of magnitude, while the parallel implementation gives an additional 4 to 6 times speed improvements.

PIECEWISE LINEAR APPROACH FOR TIMING SIMULATION OF VLSI CIRCUITS ON SERIAL AND PARALLEL COMPUTERS

BY

ONGKY TEJAYADI

B.S., University of Illinois, 1981
M.S., University of Illinois, 1983

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1988

Urbana, Illinois

PIECEWISE LINEAR APPROACH FOR TIMING SIMULATION OF VLSI CIRCUITS ON SERIAL AND PARALLEL COMPUTERS

Ongky Tejayadi, Ph.D.
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, 1988

The work presented in this thesis deals with the development of a fast and fairly accurate Computer Aided Design software for simulating very-large-scale-integrated (VLSI) circuits. The methods rely on piecewise linearized nonlinear elements in the circuits.

The piecewise linear approaches explored in this work are

1. A fast piecewise linear Gauss-Seidel waveform relaxation method.
2. A slower but more accurate piecewise linear method based on simplices.
3. A Gauss-Seidel piecewise linear method with dynamic partitioning.

Also described is a mixed method which combines the fast piecewise linear method and the dynamic partitioning method. The circuit to be analyzed is partitioned into dc-connected subcircuits and then sequenced for analysis. Small subcircuits are solved using the fast piecewise linear method while large subcircuits, including the strongly-connected components in the circuit, are solved using the dynamic partitioning method.

A parallel implementation of the Gauss-Seidel piecewise linear method with dynamic partitioning on a uniprocessor computer is studied. Algorithms for the parallel implementation of the dynamic partitioning approach on a multiprocessor with shared memory (Alliant FX/8) are also explained in detail.

The piecewise linear methods presented in this work have been implemented in a set of programs called PLATINUM. The waveforms generated by PLATINUM are fairly accurate as compared to those for SPICE2, and the speedup for a uniprocessor machine is over two orders of magnitude, while the parallel implementation gives an additional 4 to 6 times speed improvements.

ACKNOWLEDGEMENTS

First I would like to thank my advisor, Professor Ibrahim N. Hajj, who gave me a great deal of assistance. I am appreciative of the opportunity to work at the Digital and Analog Circuit Group at the Coordinated Science Laboratory. I would also like to thank Professors Timothy N. Trick, Vasant B. Rao and Ahmed Sameh for serving on my dissertation committee. My special thanks go to D. Smart, T. K. Yu and others of the Digital and Analog Circuit Group for their friendship and technical discussions.

Also, I would like to acknowledge those who have supported me at the University of Illinois, namely, the Semiconductor Research Corporation, the Joint Services Electronics Program, and the Center for Supercomputer Research and Development at the University of Illinois.

Finally, I would like to thank my parents and my sister who provided me with their love and support. I have come this far because of them.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Fabrication of integrated circuits is expensive and errors encountered after the process is completed cannot be corrected. Therefore, before the circuit is fabricated, it is important to design the circuits as best as possible and then simulate the operation of the circuits to check if the performance matches the specifications. In general, simulation can be divided into classes corresponding to the different levels of the design:

1. functional level simulation
2. register transfer level simulation
3. logic simulation
4. timing simulation
5. circuit simulation
6. device simulation
7. process simulation

Although simulation at each of these levels is important for successful design, this work is aimed at developing fast and reliable methods for circuit and timing simulations of large-scale circuits. Before describing the new methods, well-established techniques as well as recently proposed ones are briefly reviewed below.

Circuit simulators, such as SPICE2 [1] and ASTAP [30], provide accurate results. These standard circuit simulators basically follow the procedure indi-

cated below :

1. Transform the nonlinear differential-algebraic equations describing the dynamic behavior of the circuit into nonlinear algebraic equations using implicit integration methods.
2. Generate linear equations by iteratively applying the Newton-Raphson formula to the nonlinear algebraic equations.
3. Solve the linear equations at each Newton-Raphson iteration using sparse Gaussian elimination techniques.

More recent circuit simulators apply tearing or partitioning methods to lower the computation time. Tearing refers to breaking the original system into subsystems, solving each subsystem separately, and then taking care of the interconnections among them. The main advantage of dividing the original network into subnetworks is that the inactivity or latency of the subcircuits can be exploited. It has been observed that inactivity or latency in large digital circuits accounts for up to 80 percent of the network variables. The numerical convergence and stability properties of tearing methods are the same as those of standard circuit simulators, provided direct methods are used to solve the partitioned equations. One well-known tearing method is equivalent to reordering the system variables into bordered block diagonal (BBD) form [51]

$$\begin{bmatrix} D & P \\ Q^T & T \end{bmatrix} \begin{bmatrix} \mathrm{v} \\ \mathrm{w} \end{bmatrix} = \begin{bmatrix} \mathrm{y} \\ \mathrm{s} \end{bmatrix}$$

where $\mathrm{w} \in R^k$ is the vector of tearing variables and $\mathrm{v} \in R^m$ is the vector of the rest of the variables, T is a kxk tearing matrix, and D is an mxm block diagonal matrix. The tearing variables w are solved by eliminating variables v

$$(T - Q^T D^{-1} P)\mathbf{w} = \mathbf{s} - Q^T D^{-1}\mathbf{y}$$

Then the rest of the variables v are solved :

$$D_i \mathbf{v}_i = \mathbf{y}_i - P_i \mathbf{w}$$

where the subscript i indicates the i^{th} subcircuit.

One example of a circuit simulator that uses the above tearing method is SLATE [2]. Due to the fact that only a small percentage of the total subnetworks are active at a particular time, and hence only few subnetworks need to be analyzed, this method can provide savings in computation time.

Another way to save computation time is to apply a relaxation based solution method [11], which can also be considered as a form of tearing. An example of a circuit simulator that utilizes a relaxation-method is RELAX [11], which solves the equations at the nonlinear, algebraic-differential equation level. In RELAX [11], while solving for unknown variables assigned to each subsystem for the time period $[t_i, t_j]$, the rest of the unknown variables not assigned to that particular subsystem are relaxed to waveforms of previous iterations. The advantage inherent in the waveform relaxation method is that each subsystem can be solved using its own time step, and thus can exploit latency in a natural way. The main disadvantage is that for subsystems with strong coupling among them the method converges very slowly.

Another way to reduce simulation time is to use timing simulators, switched-level simulators [3,10], or timing verifiers [14,15]. Timing simulators use methods similar to those used in circuit simulators, while switched-level simulators and timing verifiers use approaches that are completely different. The speed and accuracy of these simulators cover a broad range; in general,

switched-level simulators and timing verifiers are faster than the circuit-oriented timing simulators; however, switched-level simulators and timing verifiers are less accurate. Some examples of timing simulators that use approaches similar to those of circuit simulators are MOTIS [5], MOTIS-C [6], MOTIS II [7], SPLICE [41] and PREMOS [8]. To reduce the computation time at each time point a one-sweep Gauss-Jacobi method is used in MOTIS [5] and a one-sweep Gauss-Seidel approach in MOTIS-C [6]; i.e., the iteration is not carried out until convergence. In SPLICE [41] the relaxation method is applied to the nonlinear difference equations. It is similar to MOTIS-C [6] except that the iterations are carried out until convergence or until the number of iterations exceeds some predetermined value. In the latter case the time step is reduced and the calculation is repeated. The iterations are performed to achieve accuracy and convergence. PREMOS [8] applies a Gauss-Seidel method similar to the one used in MOTIS-C [6], except that the unknown variables in the Gauss-Seidel formulation are predicted based on previous values.

Switched-level simulators are somewhat related to logic simulators in that they use levels defined as 0, 1 and X (X is the undefined or unknown level). Nodes in a circuit are assigned strengths which determine if the nodes can affect or be affected by other nodes. Each transistor in the circuit is assigned a state. During the analysis the states of the transistors are first held fixed and the nodes are updated; the transistor states are then modified while the node states are kept fixed. An example of simulators that use this procedure is MOSSIM [3]. Another approach to switched-level simulation is presented in [4] and is implemented in the simulator EXPRESS. The method relies on the evaluation of

symbolic logic expressions which are generated automatically by the simulator. The method is also able to handle faults injected into the circuits. Another type of switched-level simulator is MOSTIM [10]. In this case, the third level X represents a state that is above a chosen low-threshold level and below another chosen high-threshold level. Note that this level contains timing information while the X state of the other switched-level simulators (MOSSIM , EXPRESS) only represents undefined or unknown values, which means that the X level can also be 0 or 1. In MOSTIM, delay tables for a basic inverter circuit and for an inverter with transmission gate are constructed using circuit simulation runs in a preprocessing step. Delay information is then extracted from these runs. The delays of nonstandard primitives are obtained from the tables by using scaling of existing primitives. The simulator is in many cases over two orders of magnitude faster than SPICE, and the X level provides fairly accurate timing information. One drawback is that the tables require a large memory space and have to be constructed for each technology.

Timing verifiers, on the other hand, determine the timing of critical paths in a circuit. Timing verifiers use methods that are signal-value independent. However, timing verifiers may report false critical paths, or paths that are never activated in reality. To handle this weakness some mechanisms are incorporated by the timing verification programs. Two examples of timing verifiers are Crystal [14] and TV [15]. The difference between the two is that Crystal employs a depth-first search in determining the critical paths, while TV uses a breadth-first search. The timing or delay calculation of the critical path is based on approximating the transistors by linear resistors and then determin-

ing the dynamics of the resulting RC network based on some RC time constant approximation, such as the one suggested by Penfield and Rubinstein [21].

In this study a new method for fast timing **simulation** based on piecewise linearized transistor models is developed. The method has computational speed comparable to that of switch-level timing simulation, and at the same time produces waveforms close to those produced by standard circuit simulation. The use of piecewise linear (*pwl*) techniques for time-domain analysis of electronic circuits is not new [32]. It has been used by Hajj and Skelboe in [12], where the numerical properties of implicit integration formulas are analyzed when applied to the solution of *pwl* systems without partitioning. In [28] Laplace transform techniques are applied to compute the solution in the linear regions of the *pwl* equations. In [13] Kaye and Sangiovanni-Vincentelli use Laplace transforms and Gauss-Jacobi method to compute the solutions of *pwl* systems of equations, where the set of equations is partitioned into systems of scalar equations. A major time-consuming step when applying the Laplace transform method to the solution of *pwl* equations is the computation of the intersection of the solution trajectories with the region boundaries. In [37] a Gauss-Seidel technique is used to solve *pwl* circuits. In this case the circuit partitions are fixed; in addition, Gauss-Seidel techniques are used to solve the *pwl* equations within each partition. The method can thus be too slow when strong coupling exists among the circuit variables.

More recently there have been a few papers dealing with methods related in some respects to *pwl* techniques, most notably Elogic [16,17] and Cinnamon [18], related in the sense that linear or *pwl* transistor models are used. These

two simulators will be described next.

In Elogic [16.17] the transistor model used is the small-signal model linearized at the operating point, or line-thru-origin model. An n-dimensional table consisting of a Norton equivalent circuit for each output node as a function of controlling voltage states is constructed. Unlike the method applied in a conventional simulator, Elogic discretizes the voltage level, calculates the total conductance and total current at each node, and determines the time when the next discretized voltage level is crossed. The time increment Δ t is computed as follows :

$$\Delta t = (C_N x \Delta V)/(I_N - (V_N x G_N))$$

where C_N is the capacitor at node N , I_N is the total current at node N , V_N is the voltage at node N at the present time point , and G_N is the total conductance obtained from the table for node N. Only transitions between adjacent states are allowed by Elogic. Since waveform relaxation iteration is not carried out until convergence, Elogic might make a wrong transition to a new voltage state. The solution to this problem is to use small voltage steps. A better version (Elogic2) which applies the trapezoidal method for discretizing the time derivative and solves strongly coupled nodes together was developed. Solving strongly coupled nodes together eliminates the nonconvergence problem of the waveform relaxation method as applied in Elogic1. Since it is more expensive to use Elogic2, the program is used during analysis only when Elogic1 fails.

Cinnamon uses a method similar to the one used in Elogic in that the voltage level is discretized. However, the transistors are linearized at each time a discretized voltage level is crossed (at each "event"), rather than obtaining the

transistor model information from tables. The time when a voltage level is crossed is determined by approximating the solution obtained using the Laplace transform method. The approximation is that if the amplitude of the exponential term corresponding to the smallest (absolute value) of the system eigenvalues is smaller than the voltage step ΔV, then this term is the dominating term of the solution. This method of solution gives more accurate results than the approach of Elogic, but the use of the Laplace transform method could slow down the solution process.

There are three *pwl* approximation methods described in this study. The three methods construct *pwl* models at the outset in a preprocessing step of the simulation - as is done in Hajj and Skelboe [12] and Kaye and Sangiovanni [13], so it is not necessary to linearize frequently as is done in Cinnamon. Compared to the tables for the transistor models used in Elogic, the table sizes in our approach is smaller, since the tables are one-dimensional, and fewer breakpoints are needed.

The first method is a modification of the Chien and Kuh method of performing *pwl* analysis on simplices [40]. In the original method there is no implication of piecewise linearizing the network elements, but rather the method is applied to general *pwl* functions. In our case both the network elements and the solution curve are piecewise linearized. There are some advantages to using this method. It is simpler than the more common Katzenelson method [29], in that there is no need to explicitly calculate the boundary crossings when the solution curve enters a new *pwl* region. Moreover, there is no need for the function to have a derivative; that is, it is not necessary to con-

struct the Jacobian matrix as is done in the Newton-Raphson method. In fact, the function describing the device characteristics need not be known. Only data points on the current-voltage characteristic curves are needed. In addition, the convergence of the method is the same as that of Katzenelson's method.

The second method combines a fast *pwl* method and the waveform relaxation approach. This method is based on the work of Hajj and Jung [39]. The idea is to partition the system into a set of scalar *pwl* dynamic equations, solve each equation by inspection, and iterate using the Gauss-Seidel waveform relaxation approach until convergence. It is found, however, that for strongly-coupled nodes the method proposed in [39] converges very slowly. Modification to the original method is described in the next chapter.

The third method is a completely novel one, which dynamically partitions the network during the analysis so that the resulting linear matrix representing the piecewise linearized circuit is as block-diagonal as possible. The dynamic partitioning involves the comparison of integers representing regions of transistor operation. Fast computation speed without much loss of accuracy has been obtained using the third approach. Another good feature of the method is the inherent parallelism of the block-diagonal form as a result of the dynamic partitioning, and thus parallel processing can be efficiently used.

The *pwl* transistor model and the first and second *pwl* methods mentioned above, namely, the *pwl* method on simplices and the Gauss-Seidel *pwl* WR approach are explained in Chapter 2. Chapter 3 is devoted to dynamic partitioning methods. An implementation of the dynamic partitioning method on parallel processors is described in Chapter 4. The implementation of the

approaches for sequential and for parallel machines and some examples are given in Chapter 5. Conclusion and suggestions for future works are described in the final chapter. Modification to the *pwl* transistor model to incorporate short channel effects is described in Appendix A. A brief description of the program PLATINUM, which is an implementation of the dynamic partitioning approach, is given in Appendix B.

CHAPTER 2

PIECEWISE LINEAR SOLUTION METHODS

2.1. Introduction

Simulating entire VLSI circuits using standard circuit simulation programs such as SPICE is very time-consuming, due to the large size of the circuit. *Pwl* methods could be attractive because they simplify nonlinear model representation, and therefore, would reduce model evaluation time considerably. In addition, some *pwl* methods offer better convergence properties as compared to the standard Newton-Raphson method used in standard circuit simulators such as SPICE. In this Chapter 2 *pwl* methods, together with their advantages and drawbacks, are explained.

The currents and voltages in a circuit are governed by the following equations :

$$(KCL)\ A\,\mathrm{i}_b = 0 \tag{2.1.a}$$

$$(KVL)\ \mathrm{v}_b = A^t\,\mathrm{v}_n \tag{2.1.b}$$

$$(resistors)\ i_{n1} = f_{n1}(v_{n1}, i_{n2}) \tag{2.1.c}$$

$$(resistors)\ v_{n2} = f_{n2}(v_{n1}, i_{n2}) \tag{2.1.d}$$

$$(capacitors)\ q_c = f_c(v_c),\ i_c = \frac{dq_c}{dt} \tag{2.1.e}$$

$$(inductors)\ \Phi_l = f_l(i_l),\ v_l = d\frac{\Phi_l}{dt} \qquad (2.1.f)$$

where $\mathbf{i}_b$ is the set of currents in the b branches of the circuit, $\mathbf{v}_b$ is the set of voltages across the branches, $\mathbf{v}_n$ is the set of n node-to-datum voltages, and A is an n x b reduced incidence matrix which contains +1, -1 and 0 entries. v_{n2} and i_{n2} are voltages and currents across the resistors, q_c is the charge of the capacitors, v_c is the voltage across the capacitors, Φ_l is the flux of the inductors, and i_l is the current through the inductors. The tableau equations in (2) may be reduced to a smaller set using, for example, the modified nodal approach [46].

Since the work presented here is based on piecewise linearization of the nonlinear elements in the circuit, *pwl* modeling of nonlinear elements, represented by f_{n1}, f_{n2}, f_c and f_l in (2.1), will be explained in the following sections. Note that the functions in (2.1) include linear elements and independent sources. These elements, of course, need not be piecewise linearized.

2.2. Piecewise Linear Modeling

2.2.1. Two terminal elements

The *pwl* approximation of the nonlinear characteristic of a 2-terminal element is shown in Figure 3. In this case the *pwl* curve is characterized by a set of breakpoints. The breakpoints define region boundaries. In each region the equation is as follows:

$$y = a_i x + b_i$$

where the subscript i indicates the region number. The number of breakpoints

and their locations determine the accuracy of the *pwl* approximation with respect to the original function.

2.2.2. Multiterminal elements

In n-dimensional space the boundary between two regions is an (n-1)-dimensional hyperplane. In each region the *pwl* function is of the form

$$\mathbf{f}(\mathbf{x}) = \mathbf{J}_i \mathbf{x} + \mathbf{w}_i = \mathbf{y}$$

where $\mathbf{J}_i$ is a constant matrix and $\mathbf{w}_i$ is a constant vector. $\mathbf{J}_i$ and $\mathbf{w}_i$ are defined in each *pwl* region.

Modeling of multiterminal, nonlinear elements by *pwl* functions in general requires multidimensional tables. However, if the functions of several variables representing the terminal characteristics of a multiterminal element can be expressed as the sum of single-variable functions, or the sum of nested functions, then the *pwl* representation can be expressed in terms of a set of one-dimensional tables. This would save both storage and computation. In general, however, such a model decomposition is not necessary, since one can use simplices, as will be described in section 2.4. In the next section, we show how three-terminal elements, such as an MOS transistor, can be decomposed into an interconnection of two-terminal elements. Then each of the two-terminal elements is piecewise linearized. Each two-terminal *pwl* model can then be stored in a one-dimensional table.

2.2.3. Piecewise linear transistor model

The well-known simple equations of the channel current of an MOS transistor is as follow [38]:

Linear region:

$$I_{DS} = K(2(V_{GS}-V_T)V_{DS}-V_{DS}^2) \; ; \; 0 \leqslant V_{DS} \leqslant V_{GS}-V_T \qquad (2.2.a)$$

Saturation region:

$$I_{DS} = K(V_{GS}-V_T)^2 \; ; \; 0 \leqslant V_{GS}-V_T \leqslant V_{DS} \qquad (2.2.b)$$

$$K = \mu \epsilon_{ox} W / 2 t_{ox} L$$

$\mu =$ average surface mobility of carriers in the channel of the device

ϵ_{ox} = permittivity of the oxide

t_{ox} = thickness of oxide under gate

L = length of the channel

W = width of the channel

The V_{GS}, V_{DS}, and V_T are gate-to-source, drain-to-source, and threshold voltage, respectively. The terms K and V_T in the above equations are considered to be constants. It is clear that a *pwl* approximation of (2.2.a) requires the generation of a two dimensional table with $V_{GS}-V_T$ and V_{DS} as independent variables. Although interpolation on two or higher dimensional tables is feasible, it is much more efficient from the computational and storage points of view to have a one-dimensional tabular representation. Meyer [25] proposed the following model which transforms (2.2) into sums of functions of a single variable each :

$$I_{DS} = K(V_{GS}-V_T)^2 - K(V_{GD}-V_T)^2 \qquad (2.3.a)$$

$$= K f(V_{GS}) - K f(V_{GD})$$

$$= I_1 - I_2$$

$$\text{where } f(V_{GX}) = \begin{cases} (V_{GX} - V_T)^2 & \text{for } V_{GX} \geq V_T \\ 0 & \text{for } V_{GX} < V_T \end{cases} \qquad (2.3.b)$$

The model depicting Equation (2.3) is shown in Figure 1. The model can be transformed into an 'Ebers-Molls-type' model as shown in Figure 2. The α 's in Figure 2 are equal to unity to keep $I_{gate} = 0$.

The next step is to approximate the quadratic equations in (2.3.b) by *pwl* functions. An example of a graph of I *vs* V_{GX} and its piecewise linearized representation is shown in Figure 3. For timing analysis a three-segment model has been found to be adequate for providing acceptable accuracy. The resulting circuit, depicted in Figure 4, consists of a conductance and a current source, where the value of the conductance is the slope of the linear function in a segment and the current source is the intercept of the function with y axis (the I axis).

Using an implicit integration formula, such as the backward Euler formula, to approximate the time derivatives in (2.1e) and (2.1f), the resulting *pwl* circuit equations at time t_n are of the form

$$g(x^n) = 0 \qquad (2.4)$$

where x could be the modified nodal equation variables, and x^n the value of x at time t_n. Equation (2.4) is usually solved by using Newton's method. At every iteration in Newton's method, the linearized equations are of the form:

$$A x = b \qquad (2.5)$$

A number of iterations may be necessary before the process converges, provided

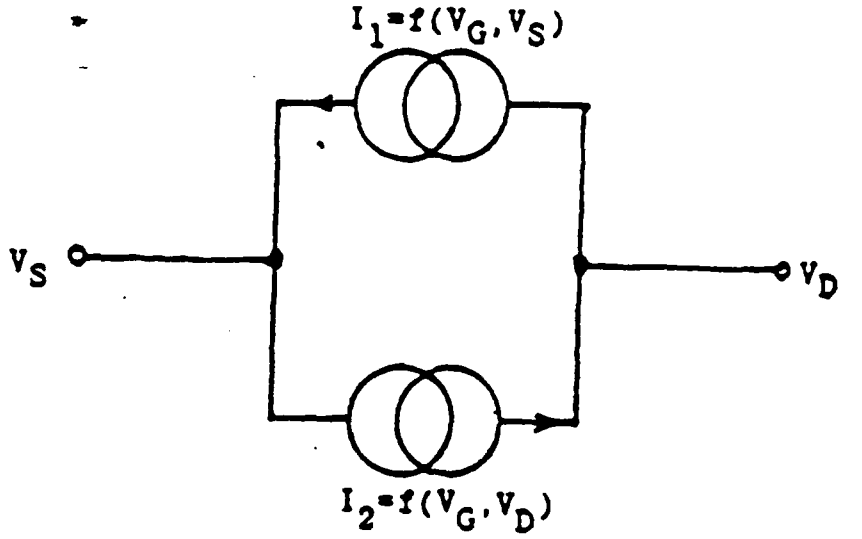


Fig. 1 Transistor model from Equation (2.2)

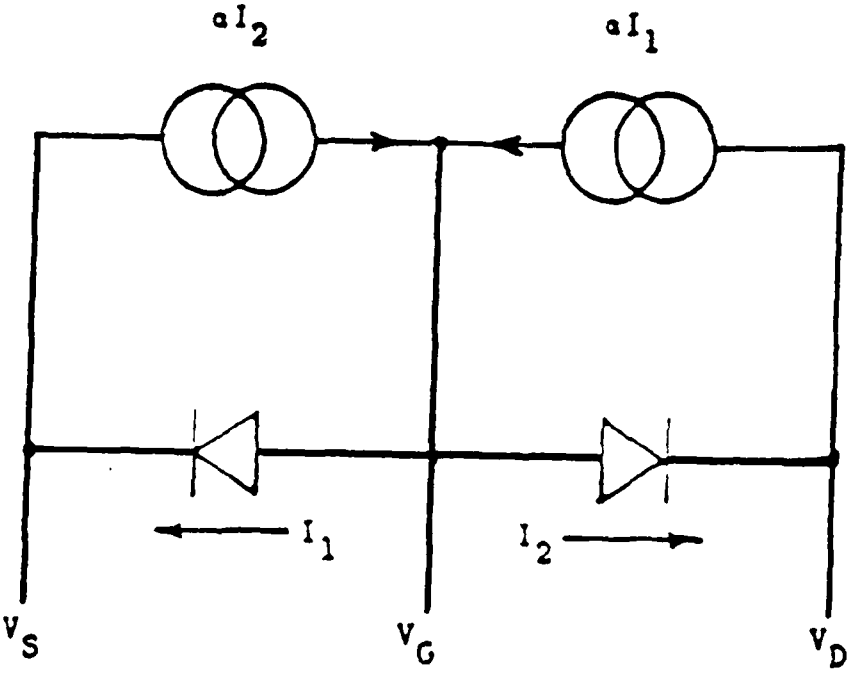


Fig. 2 "Ebers-Mollis type" model of an MOS transistor

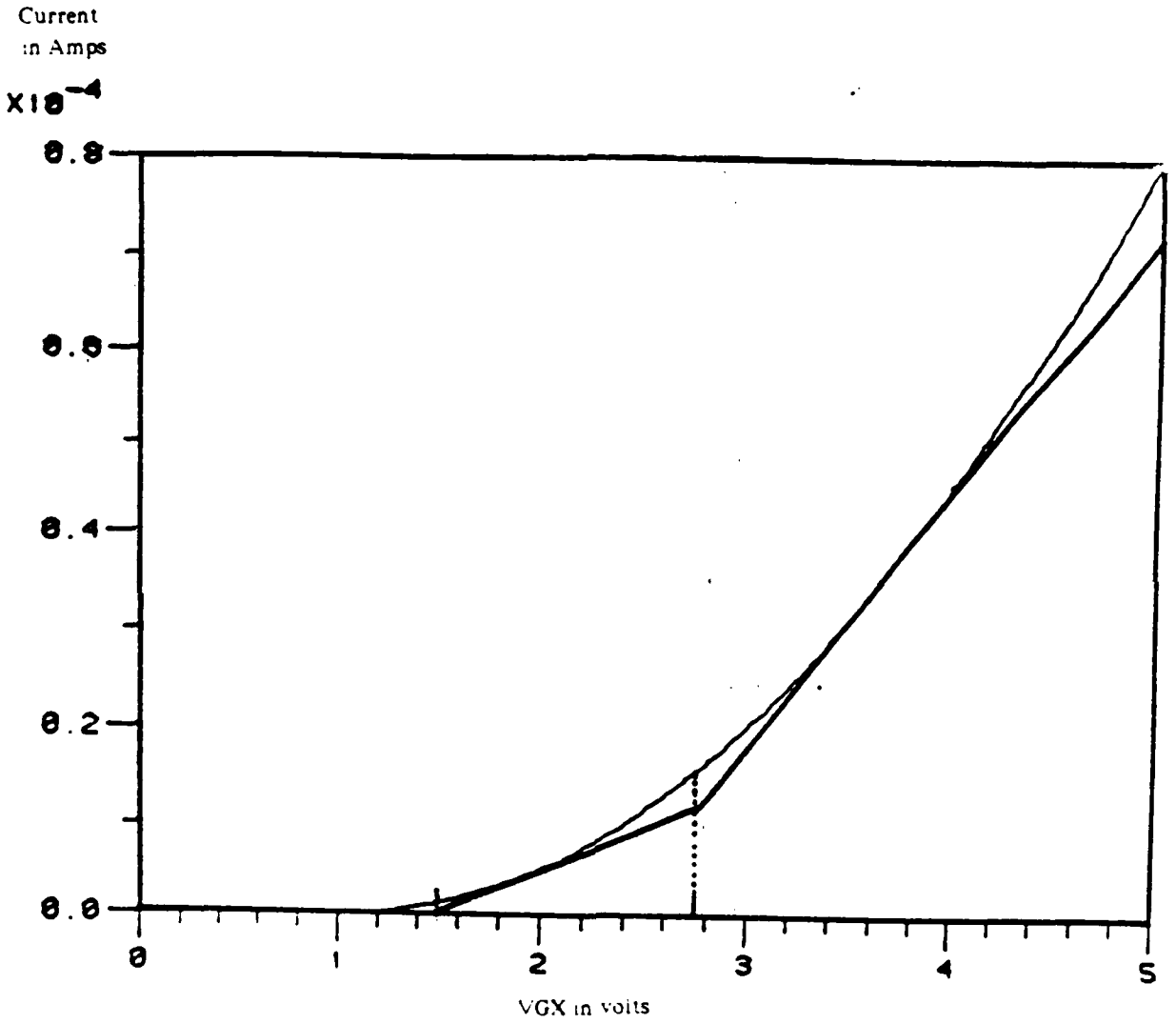
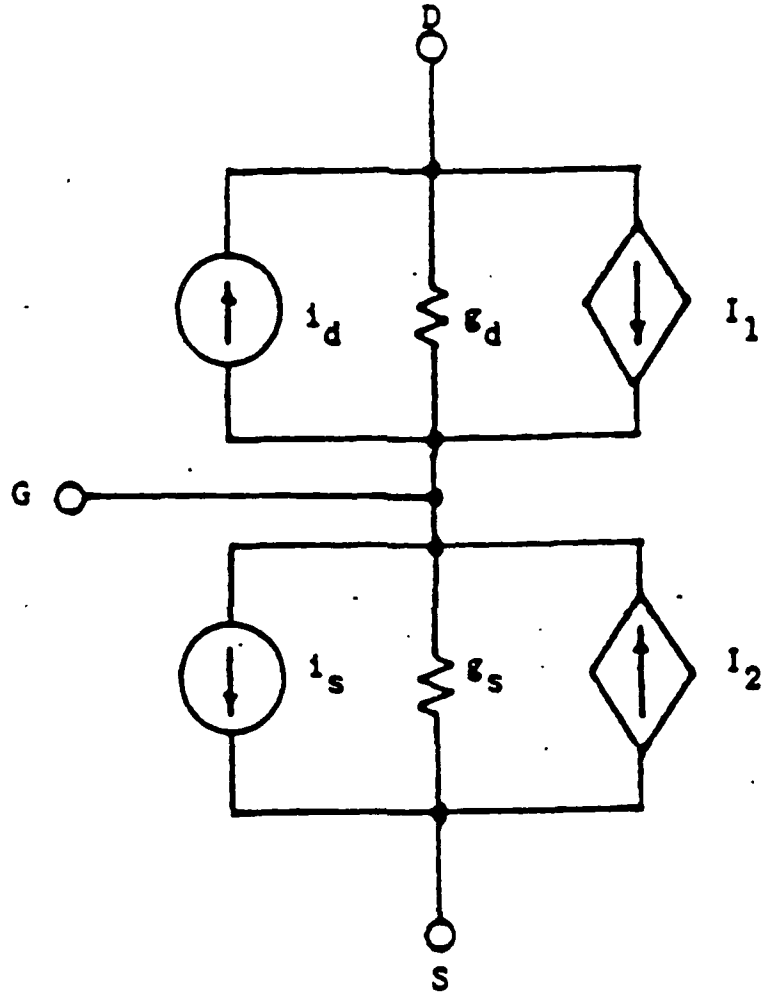


Fig. 3 The original and pwl current vs. voltage plots



$$I_1 = V_{GS} * g_s + i_s$$

$$I_2 = V_{GD} * g_d + i_d$$

Fig. 4 A piecewise linear transistor model

it does converge. The matrix A, which is usually sparse in circuit analysis, is solved by sparse matrix solution methods to reduce the computational burden.

A modified Newton's method for *pwl* equations, known as the Katzenelson's method, guarantees convergence. In Katzenelson's method the next iteration point is chosen to be the intersection of the solution trajectory with the boundary hyperplane unless the solution is found within the region. A drawback of Katzenelson's method is the time it needs to determine boundary crossings. A variant of Katzenelson's method, the *pwl* method on simplices, finds the boundary crossings in a simple and more efficient way. The *pwl* method on simplices is explained next.

2.3. Piecewise Linear Approach on Simplices

This method was first proposed by Chien and Kuh [40]. It is conceptually similar to the well-known Katzenelson method. The advantages of this method are as follows:

1. There is no need to determine boundary crossings as is done in the Katzenelson method. Instead, a vertex replacement is performed on simplices.
2. There is no need to calculate the Jacobian matrix as is required in the Newton-Raphson formula. In this sense the method is more general since a function which does not have a derivative can still be solved.
3. The functions describing the current-voltage and charge-voltage characteristics need not be known. Sample points on the multidimensional characteristics are sufficient for the computation. This implies that

new devices based on new technologies can be studied without having the function governing the operation of the device derived.

In the next few paragraphs the definition of a simplex is reviewed. This is followed by a description of the Chien and Kuh method. Finally, an algorithm based on the approach is presented.

Let $x_0, \cdots, x_n \in R^n$. A simplex, known also as a closed convex hull, $S(x_0, \cdots, x_n)$ is defined by

$$S(x_0, \cdots, x_n) =$$

$$\left\{ x \in R^n \mid x = \sum_{i=0}^{n} \mu_i x_i, 0 \leq \mu_i \leq 1, i = 0,1,2,\ldots,n \text{ and } \sum_{i=0}^{n} \mu_i = 1 \right\}$$

$x_0, \cdots, x_n$ are called the vertices of the simplex $S(x_0, \cdots, x_n)$. A simplex $S(x_0, \cdots, x_n)$ is called proper if and only if the $(n-1) \times (n-1)$ matrix

$$\begin{bmatrix} x_0 & \cdots & x_n \\ 1 & \cdots & 1 \end{bmatrix}$$

is nonsingular.

The boundary H_k corresponding to the vertex x_k is defined as

$$H_k = \left\{ x \in R^n \mid x = \sum_{i \neq k} \mu_i x \right\}$$

As will be explained later in the chapter, this definition of boundary is very useful in determining where the solution curve should go. Due to the fact that there is a one-to-one correspondence between a boundary and a vertex, instead of determining which boundary is to be crossed by the solution curve, the corresponding vertex to be removed is determined. This vertex removal turns out to be simpler in calculation and programming than finding the

boundary crossing. Reference [40] contains a complete explanation and derivation of the method. The following paragraphs describe the idea and the algorithm.

Let the original function to be piecewise linearized be $\mathbf{f}(\mathbf{x}) = \mathbf{y}$ where $\mathbf{f}(.) : R^n \rightarrow R^n$. A function $\mathbf{g}(.)$ approximating the original function $\mathbf{f}(.)$ on $S(\mathbf{x}_0, \cdots, \mathbf{x}_n)$ is defined by

$$\mathbf{g}(\mathbf{x}) = [\mathbf{f}(\mathbf{x}_0), \cdots, \mathbf{f}(\mathbf{x}_n)]\,\boldsymbol{\mu}$$

for $S(\mathbf{x}_0, \cdots, \mathbf{x}_n)$ and $\boldsymbol{\mu} = [\mu_0, \mu_1, \cdots, \mu_n]^T$ defined previously for the representation of $\mathbf{x} \in S(\mathbf{x}_0, \cdots, \mathbf{x}_n)$.

In summary, the representation of a point $\mathbf{x}$ in a simplex $S(\mathbf{x}_0, \cdots, \mathbf{x}_n)$ and the *pwl* function $\mathbf{g}(\mathbf{x})$ are as follows :

$$\mathbf{x} \in S(\mathbf{x}_0, \cdots, \mathbf{x}_n)$$

$$\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_0 & \cdots & \mathbf{x}_n \\ 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{\mu} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{f}(\mathbf{x}_0) & \cdots & \mathbf{f}(\mathbf{x}_n) \\ 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{\mu} \end{bmatrix}$$

Once the boundary to be crossed is identified, then one needs to determine the new simplex entered. Since corner crossing is not allowed, all vertices, except one, remain the same. It is shown in [40] that the new vertex $\hat{\mathbf{x}}_k$ is the combination of the old value $\mathbf{x}_k$ and two of its adjacent vertices; that is,

$$\hat{\mathbf{x}}_k = \mathbf{x}_{k+1} + \mathbf{x}_{k-1} - \mathbf{x}_k$$

where k indicates the position of the altered vertex.

The solution algorithm given below is a slightly modified version of the one described in [40].

Step 1 :

Choose

x_0 and $x_i = x_{i-1}+E_i$, $i=1,2,....,n$ *where* $E_i = [0,....0,e_i,0,....0]^T$

and $e_i > 0$ is the i th component of E_i

Step 2 :

Let $\mu^0 = [1,....,1]^T/(n+1)$; that is, $x^0 = \frac{1}{(n+1)}\sum_{i=0}^{n} x_i$ (x^0 is the

center of the initial simplex).

Set i=0

Step 3 :

Compute $\hat{\mu}^i$ according to the equation

$$\begin{bmatrix} f(x_0) & \dots & f(x_n) \\ 1 & \dots & 1 \end{bmatrix} \hat{\mu}^i = \begin{bmatrix} y^* \\ 1 \end{bmatrix}$$

If every component of $\hat{\mu}^i$ is non-negative, a solution is found

$$x^* = [x_0, \dots, x_n]\hat{\mu}^i \quad . \quad STOP$$

Step 4 :

Otherwise, compute λ^i from

$$\mu(\lambda^i) = \mu^i + sgn(\hat{\mu}^i_k)\lambda^i(\mu^i - \hat{\mu}^i)$$

such that

i) $0 \leq \mu(\lambda^i) \leq 1$ for $0 \leq \lambda \leq \lambda^i$

ii) there exists one and only one index k satisfying $\mu(\lambda^i)_k = 0$

iii) $1 > \mu(\lambda^i)_j > 0$ for $j \neq k$

In practice :

Find minimum t ($tmin$) from

$0 = \mu_k^i + t(\mu_k^i - \hat{\mu}_k^i)$; if$(\mu_k^i - \hat{\mu}_k^i) < 0$

- or - from

$0 = \mu_k^i - t(\mu_k^i - \hat{\mu}_k^i)$; if$(\mu_k^i - \hat{\mu}_k^i) > 0$

Then calculate

$\mu(tmin) = \mu^i \pm tmin(\mu^i - \hat{\mu}^i)$

Step 5 :

Replace $\hat{x}_k$ by $(x_{k+1} + x_{k-1} - x_k)$

Let i=i+1 and go to Step 3.

We found that the method is slow in analyzing circuits. To reduce the computation time the nonlinear network elements are piecewise linearized and tabulated. As a result, we piecewise linearize two things: one is the network elements and second is the solution space which becomes the space of simplices. The piecewise linearization of the network elements is not proposed in the original idea given in [40]. A variable time-step method described in Wei's thesis [43] is used. A 10-stage chain of inverters analyzed using this method requires about 20 seconds of CPU time while SPICE needs about 13 seconds.

Parallel implementation of the method could reduce the computation time. Each vertex of the simplex consists of a set of numbers representing a set of voltages. For example, an inverter with a pass transistor is represented by a simplex with 3 vertices. Each vertex consists of 2 numbers, representing the 2 voltages in the circuit. Each vertex, which is a column in the matrix of step 3 of the algorithm, can be solved in parallel. Because each vertex provides a

complete set of voltages, the entries of the corresponding column can be computed concurrently. This is performed until all the columns of the matrix are calculated.

The results of the implementation of the method on the Alliant FX/8, a vector-parallel computer with shared memory, were found to be discouraging. The speedup was only a factor of less than two as compared to SPICE. Although parallelization of the matrix entries is possible, the resulting matrix is dense, and therefore, no sparse matrix technique can be applied to reduce computation. As the circuit becomes larger, the calculation of the dense matrix could become prohibitive.

2.4. Relaxation Methods

The circuit analysis method described so far solves (2.4) as well as (2.5) directly; i.e., no relaxation is used. Alternatively, relaxation techniques could be used to solve (2.5) (e.g., linear Gauss-Seidel or Gauss-Jacobi) or (2.4) (nonlinear Gauss-Seidel or Gauss-Jacobi). In these methods the time step is controlled at the global circuit level, and thus are referred to as pointwise relaxation methods. The pointwise Gauss-Seidel method of solving (2.4) is as follows:

repeat { **foreach** (j **in** N {

$$\textit{solve } g_j(x_1^{k+1}, \ldots, x_j^{k+1}, \ldots, x_N^{k}) = 0 \text{ for } x_j^{k+1}; \} \} \tag{2.6}$$

until ($\| \mathbf{x}^{k+1} - \mathbf{x}^{k} \| \leq \epsilon$)

The **foreach** implies that the computation for each value j in the ordered set N must proceed sequentially and in the order specified by the set.

The pointwise Gauss-Jacobi method of solving (2.4) is

repeat { **forall** (j **in** N {

$$solve\ g_j(x_1^k, \ldots, x_j^{k+1}, \ldots, x_N^k) = 0 \text{ for } x_j^{k+1} : \} \} \tag{2.7}$$

until ($\| x^{k+1} - x^k \| \leq \epsilon$)

The **forall** implies that the computation for all values of j in the ordered set N may proceed concurrently, i.e., in parallel and in any order.

Relaxation techniques can also be applied at the differential equation level; i.e., each subcircuit can be solved using its own time step. The Gauss-Seidel waveform relaxation method of solving a system of nonlinear differential equations of the form

$$\dot{x} = f(x, t) \tag{2.8}$$

is

$$\dot{x}_i^{n+1} = f_i(x_1^{n+1}, \ldots, x_{i-1}^{n+1}, x_i^{n+1}, x_{i+1}^{n}, \cdots, x_m^{n}) \tag{2.9}$$

$$x_i^{n+1}(0) = x_i^n(0)$$

while Gauss-Jacobi waveform relaxation method is

$$\dot{x}_i^{n+1} = f_i(x_1^{n}, \ldots, x_{i-1}^{n}, x_i^{n+1}, x_{i+1}^{n}, \cdots, x_m^{n}) \tag{2.10}$$

$$x_i^{n+1}(0) = x_i^n(0)$$

The vector x_i in the above equation corresponds to the variables in the subcircuit i. In the waveform relaxation method the subcircuit variables are solved for a time window T. In MOS circuits, subcircuits are often obtained by partitioning the circuit into dc-connected components. Floating capacitors such as gate-source and gate-drain capacitors cause local feedback among subcircuits. In timing analysis these small floating capacitors are replaced by equivalent capacitances from the nodes to the ground [27]. As a result, the local feedback paths

among the subcircuits caused by the floating capacitors are eliminated. When applying the Gauss-Seidel method, sequencing the subcircuits for analysis could reduce the computation time. In the following subsection, sequencing of the subcircuits for analysis is described.

2.5. Analysis Sequencing

Analysis sequencing is applied after the circuit is partitioned. While partitioning and sequencing involve some overhead, the overall result is a reduced computation time. The idea is that if it is possible to partition the circuit into "one-way" subcircuits, then only one sweep of Gauss-Seidel analysis is needed for solving the circuit.

A circuit which has been partitioned into dc-connected subcircuits can be represented by a directed graph G(V,E) where V is a set of vertices representing subcircuits and E is a set of edges depicting signal lines from fanout to fanin. In the circuit, an edge $e \in E$ with an arrow from vertex x to vertex y is the result of dependent current sources due to MOS transistors in subcircuit Y. The following definitions about graphs will be used in the description of the sequencing algorithms.

Definition A :

Given a vertex v of G(V,E), the set of fanin vertices and fanout vertices of vertex v

are :

$$fin(v) = \{ w \in V \mid (w,v) \in E \}$$

$$fout(v) = \{ w \in V \mid (v,w) \in E \}$$

where (x,y) denotes an edge from vertex x to vertex y. The number of fanin and fanout vertices of v are denoted by nfin(v) and nfout(v), respectively. In the following definitions and theorems we will consider ordering or sequencing the vertices of graph G(V,E) when the graph G(V,E) does not contain any feedback. This case arises in combinational circuits consisting of simple transistor models.

Definition B :

Vertex v_i in G(V,E) is a predecessor of vertex v_j if and only if there is a directed edge from v_i v_j .

If v_i is a predecessor of v_j, then v_j is a successor of v_i.

Definition C :

A linear ordering or sequencing is called a topological order if for every predecessor v_i of v_j in the graph G(V,E), the v_i *precedes* v_j in the linear ordering.

Theorem a [31] :

The vertices in a directed graph can be arranged in a topological order if and only if the directed graph is acyclic.

The theorem implies that for any combinational circuit the graph representing the circuit is acyclic and, therefore, the subcircuits can be arranged in the topological orders. One realizes that many circuits contain feedbacks, and therefore, the corresponding graph is cyclic. The parts of the graph that contain feedback edges (known as the strongly connected component or scc) are detected using Depth-First Search Techniques. Each strongly connected

component is replaced by one new node. After the replacement the resulting new graph G' is acyclic and the sequencing method for an acyclic graph can be applied. The Tarjan's Depth-First Search algorithm [24] to find strongly connected component is as follows :

Step 1 :

(initialization step) Mark all the edges "unused." For every $v \in V$ let $k(v) \leftarrow 0$ and $f(v)$ be undefined. { $f(v) \equiv$ father of v }.

Empty S { S is a stack that stores the vertices in the order in which they are discovered }.

Let $i \leftarrow 0$ and $v = s$ { s is the 0 node or source node }.

Step 2 :

$i \leftarrow i-1$, $k(v) \leftarrow i$, $L(v) = i$ and put v on S.

Step 3 :

If there are no "unused" incident edges from v, go to Step 6.

Step 4 :

Choose an "unused" edge $v \rightarrow u$. Mark the edge e "used."

Step 5 :

(i) If $k(u)=0$, then $f(u)=v$, $v \leftarrow u$. Go to Step 2.

(ii) If $k(u) > k(v)$ (e is a forward edge). Go to Step 3.

(iii) If $k(u) < k(v)$ and if u is not on S (u and v do not belong to the same component). Go to Step 3.

(iv) If $k(u) < k(v)$ and if both vertices are in the same component (that is, u is in S), let $L(v) = \min \{ L(v), k(u) \}$ and go to Step 3.

Step 6 :

If L(v)=k(v), delete all the vertices from S down to and including v; these vertices form a strongly connected component.

Step 7 :

(i) If f(v) is defined, then L(f(v)) ← min{L(f(v),L(v)}, v ← f(v) and go to Step 3 ;

(ii) If f(v) is undefined and if there is a vertex u for which k(u)=0, then let v ← u and go to Step 2.

Step 8 :

If all vertices have been traced then STOP.

After the strongly connected components have been identified using the above algorithm, and each scc is replaced by one node, the resulting acyclic graph G' is levelized using the following algorithm.

Algorithm [44] (assign level to each vertex in G').

```
BEGIN
    Assign input vertices of the acyclic graph G' to level 0 ; k ← 0
L.  FOR each vertex v in level k DO
    For each vertex w  ϵ fout(v) DO
     BEGIN
      nfin(w) ← nfin(w)-1
      IF nfin(w)=0 THEN
      assign w to level k+1 ;
     END
    IF level k is not empty THEN
```

Go to L ;

$k \leftarrow k-1$

END

After the subcircuits are assigned levels using the above algorithm, they are analyzed starting from subcircuits connected to inputs (level 1) to the ones connected to the outputs.

Algorithm 3

BEGIN

$k \leftarrow 1$;

L. FOR each vertex v of G' at level k DO

time domain analysis of corresponding subcircuits ;

IF level k is not empty THEN

GO TO L ;

$k \leftarrow k-1$;

END

In many cases in digital circuits only some portion of the output nodes are of interest. Each one of these nodes is sometimes affected by only a small portion of the subcircuits. This implies that only some subcircuits are needed to be analyzed even if in reality the rest of the circuits are active. Since only some parts of the system are analyzed, the computation time is reduced. The method applied to take advantage of this fact is known in other areas as "back-chaining." Basically starting at the back end of the graph (that is the output

nodes of interest) one traces back until reaching the front end of the graph (the input nodes). The vertices traced during the process are the subcircuits that need to be analyzed. A typical algorithm that performs this back-chaining task is given in [43].

There are three ways of solving the strongly connected components. The obvious one is to solve them as one block. The problem with this approach is that the block might be too large. For instance, when the feedback connections are from subcircuits at the back end to the ones at the front end then the strongly connected components are practically the entire circuit, and if the circuit is large then the block may be too large to be solved at once. A better way is to apply a dynamic partitioning method, which will be described in the next chapter. The third solution involves breaking the strongly connected component into even smaller subcircuits. This is done by removing some edges from the scc so that the original scc becomes acyclic and then apply a relaxation-based solution method to the scc that has become acyclic.

Having described the Gauss-Seidel waveform relaxation method for circuit analysis and the piecewise linearization of transistor models, the fast *pwl* method will now be defined.

2.6. Fast Piecewise Linear Approach

Consider a circuit or a system described by *pwl* continuous equations of the form

$$\dot{x} = f(x(t), y(t)) = A_m x(t) + w_m + y(t), \quad m=1,2,\ldots,r \qquad (2.11)$$

where $x(.), y(.) : [0,T] \rightarrow R^n$ where R^n is divided by hyperplanes into r polyhedral regions. A_m is a constant nxn matrix and w_m a constant n vector defined for each region m.

Kaye and Sangiovanni-Vincentelli [13] use Laplace transforms and the Gauss-Jacobi method to compute the solutions of the *pwl* systems of equations. The set of equations is partitioned into systems of scalar equations. A drawback to applying the Laplace transform method to the solution of *pwl* equations is the time-consuming effort of computing the intersection of the solution trajectories with the region boundaries. The method presented here is based on the work by Jung and Hajj [39]. It combines the waveform relaxation method [11] and the Gauss-Seidel iterative method to solve the piecewise linearized equations. The original method [39] suffers from a slow convergence problem when tight coupling exists between the equations. Modification to decrease the computation time is explained in the following paragraphs.

The solution of Equation (2.11) based on a Gauss-Seidel *pwl* WR algorithm [39] is as follows:

Step(0) : Set $x_i^0(t) = x_i(0), i=2,\ldots,n, t \in [t_0, t_1]$.

$0 \leq t_0 \leq t_1 \leq T$.

.
.
.

Step(k) : Solve $\dot{x}_1^k = a_{11m} x_1^k(t) + \sum_{i=2}^{n} (a_{1im} x_i^{k-1}(t)) + w_{1m} + y_1(t)$.

$x_1^k(t_0) = x_1(t_0)$ for $x_1^k(t), t \in [t_0, t_1]$

.

.

.

Solve $\dot{x}_j^k(t) = a_{jjm} x_j^k(t)$

$$+ \sum_{i=1}^{j-1} a_{jim} x_i^k(t) + \sum_{i=j+1}^{n} a_{jim} x_i^{k-1}(t) + w_{jm} + y_j(t)]$$

$$x_j^k(t_0) = x_j(t_0), \text{ for } x_j^k(t), \ t \in [t_0, t_1]$$

.

.

.

Solve $\dot{x}_n^k(t) = a_{nm} x_n^k(t)$

$$+ [\sum_{i=1}^{n-1} a_{nim} x_i^k(t) + w_{nm} + y_n(t)],$$

$$x_n^k(t_0) = x_n(t_0), \text{ for } x_n^k(t), \ t \in [t_0, t_1]$$

At each step, instead of solving n coupled differential equations, one needs to solve only n decoupled ones. The process is repeated until convergence is obtained.

Each of the above equations is of the form

$$\dot{x}_i(t) = a_{iim} x_i(t) + w_{im} + y_i(t), \ t \in [t_0, t_1]$$

The solution to this linear first-order differential equation is

$$x_i(t) = x_i(t_0) e^{a_{iim}(t-t_0)} + e^{a_{iim}t} \int_{t_0}^{t} e^{-a_{iim}\tau} (w_{im} + y_i(\tau)) d\tau$$

If $y_i(t)$ is a constant the solution can be found by inspection:

If $a_{im} \neq 0 : x_i(t) = [x_i(t_0) + (w_{im} + c_m)\, a_{im}]e^{a_{im}(t-t_0)} - (w_{im} + c_m)\, a_{im}$

If $a_{im} = 0 : x_i(t) = x_i(t_0) + (w_{im} + c_m)(t - t_0)$

In the process of finding $x_i(t):[0,T]$ the values of a_{im}, w_{im} change due to the fact that the solution trajectory moves from one region to the next. Hence it is necessary to be able to find t when a new region is entered, which can be done.

If $a_{im} \neq 0 : t = t_0 + (ln(v_m))\, a_{im}$

where $v_m = [b_1 + c_m/a_{im}]/[x_1(t_0) + c_m/a_{im}]$

If $a_{im} = 0 : t = t_0 + (b - x(t_0))/c_m^k$

The conditions for $t \geq 0$ are

1. if $a_{im} > 0$ *then* $v_m > 0$ *or*,

2. if $a_{im} < 0$ *then* $0 < v_m < 1$ or,

3. if $a_{im} = 0$ *then* $(b - x(t_0))\, c_m^k > 0$.

In general $y_i(t)$ is not a constant. However, if $y_i(t)$ is approximated by a "staircase" function, that is, $y_i(t)$ is divided into intervals and in each interval it is represented by a step input, then the solution $x_i(t)$ for each interval can be found by inspection as derived above.

As an example, consider a *pwl* inverter circuit shown in Figure 5. The transistor model and its *pwl* approximation are explained in section 2.2.3. Solving the node equation at the output node gives

$C_L \dot{V}_{out} + I_{D1} + I_{L2} - I_{L1} - I_{D2} = 0$

$C_L \dot{V}_{out} + V_{gsd}{*}gsd + isd - V_{gdd}{*}gdd - idd - V_{gsl}{*}gsl - isl$

$- V_{gdl}{*}gdl + idl = 0$

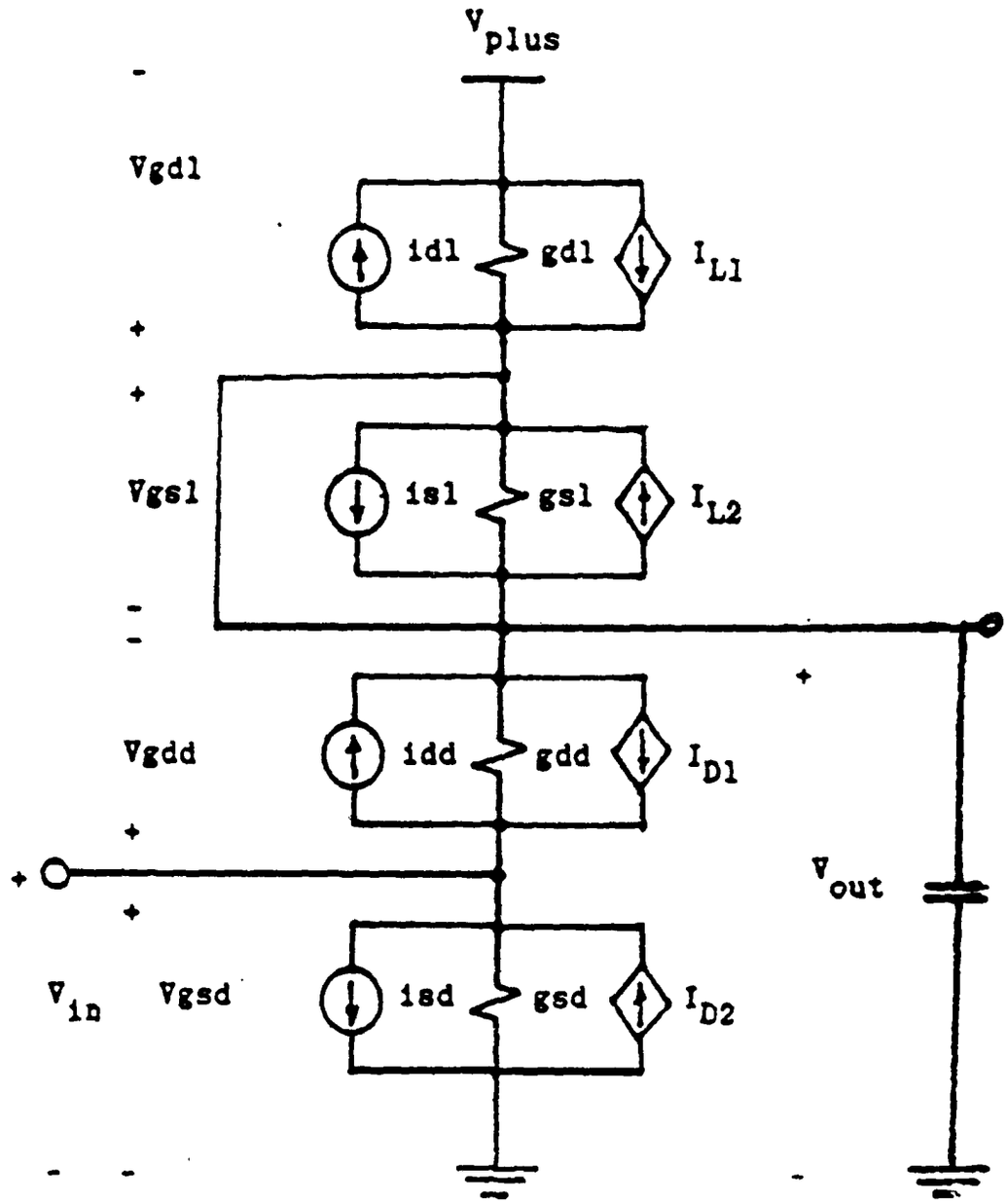


Fig. 5 Piecewise linear model of an inverter

The output is low initially and a falling step input is applied. At t=0 the input falls to zero. Checking the *pwl* model gives *isl* as the only nonzero term. The above equation becomes

$$C_l \dot{V}_{out} = isl \tag{2.12}$$

The solution to this equation is:

$$V_{out} = V_{outinit} + isl*t/C_l$$

where $V_{outinit}$ is the initial value of V_{out}.

If t is sufficiently large, then at some point in time V_{out} becomes so large that the linear model is not valid anymore. In other words, a new region in the transistor *pwl* characteristics is entered. When this happens, which in our case is at V_{out} = 3.25 volts, the time when the new region is entered is calculated using (2.12) and the *pwl* elements that are affected by V_{out} are updated. Now the gdd and idd terms are also nonzero, and the output node equation is of the form

$$C_l \dot{V}_{out} + V_{out}/R = I \ , \ R,I \neq 0$$

and the solution is

$$V_{out} = V_2 + (V_1 - V_2)*exp(-t/tc) \tag{2.13}$$

where V_2 = I*R

V_1 = initial V_{out} (=3.25)

tc = R*C_l

Again the model is only valid until another region is encountered, which in our case is at V_{out} =4. The time when the new region is entered is calculated using (2.13); the *pwl* elements are updated and the same process continues.

When V_{out} reaches 5 volts, the current sources of the load cancel each other (the driver is still off) and there is no more change to the output as long as the input does not change.

Suppose the output of this inverter is fed into another inverter. Then this output is approximated by a "staircase" function, as shown in Figure 6a (for falling waveform the approximation is shown in Figure 6b) and the same analysis as above is carried out for each time interval.

The above solution of $x_i(t)$ assumes an input time function approximated by a "staircase" function. Actually, the input can be approximated by other types of functions, for example a ramp function. However, the solution would then contain terms proportional to t and $exp(-kt)$, and so there is no simple and fast way to get the time when a new region or interval is entered.

As mentioned earlier, in some cases the method described above converges slowly. This is true for strongly coupled circuits, such as pass transistor networks, circuits with internal nodes, and circuits with floating capacitors. The reason is that an approximate waveform representation used in the waveform iterative technique does not give good convergence for strongly coupled nodes. In the following subsections various techniques to reduce the computation needed in solving coupled subcircuits and to ensure convergence are derived.

2.6.1. Pass transistor networks

From Figure 7 it is clear that the waveform at the output node depends on the other nodes. Applying the Gauss-Seidel *pwl* WR method it is found that more than 10 iterations are required for convergence, which is relatively slow.

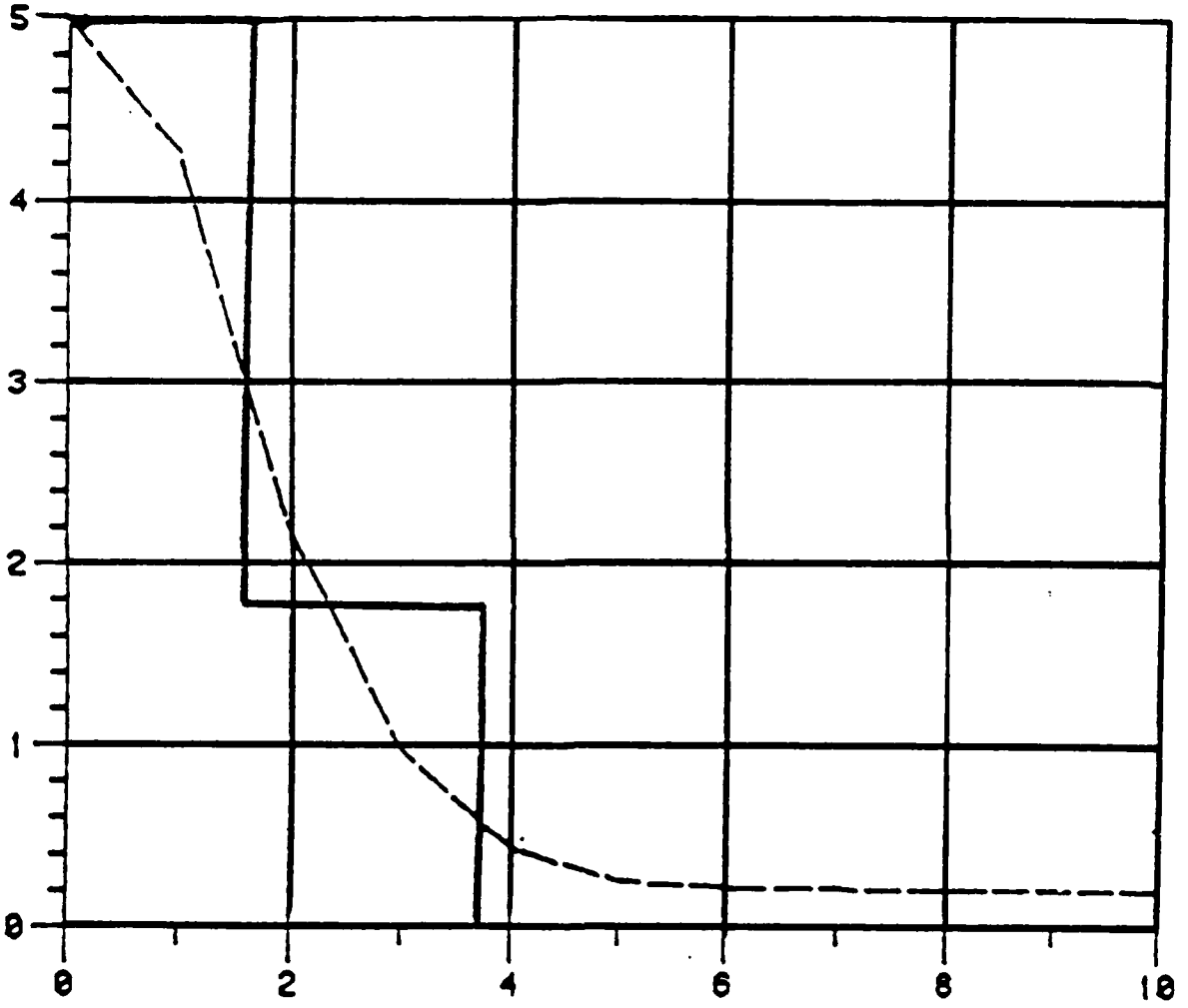


Fig. 6a Piecewise linear approximation of a falling waveform

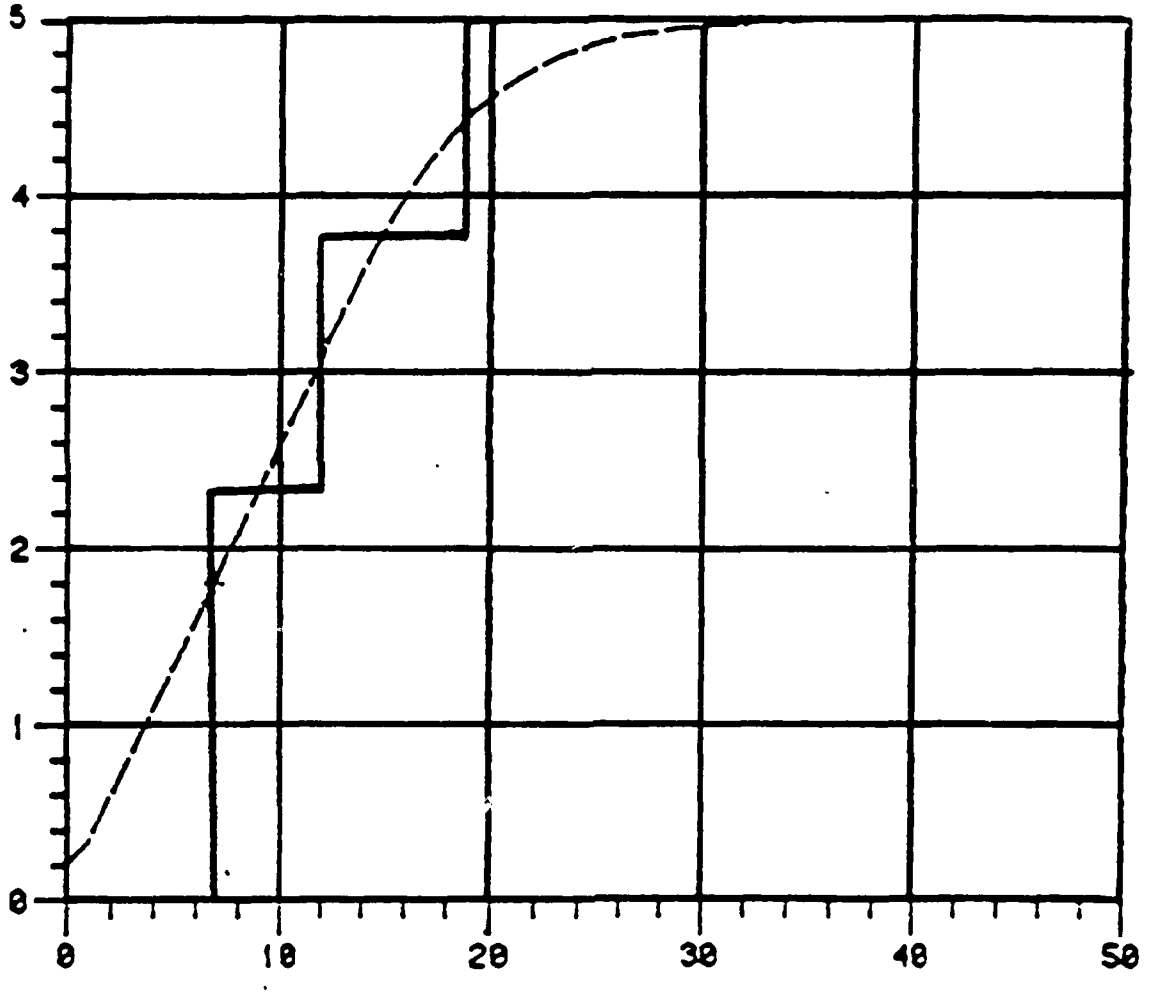


Fig. 6b Piecewise linear approximation of a rising waveform

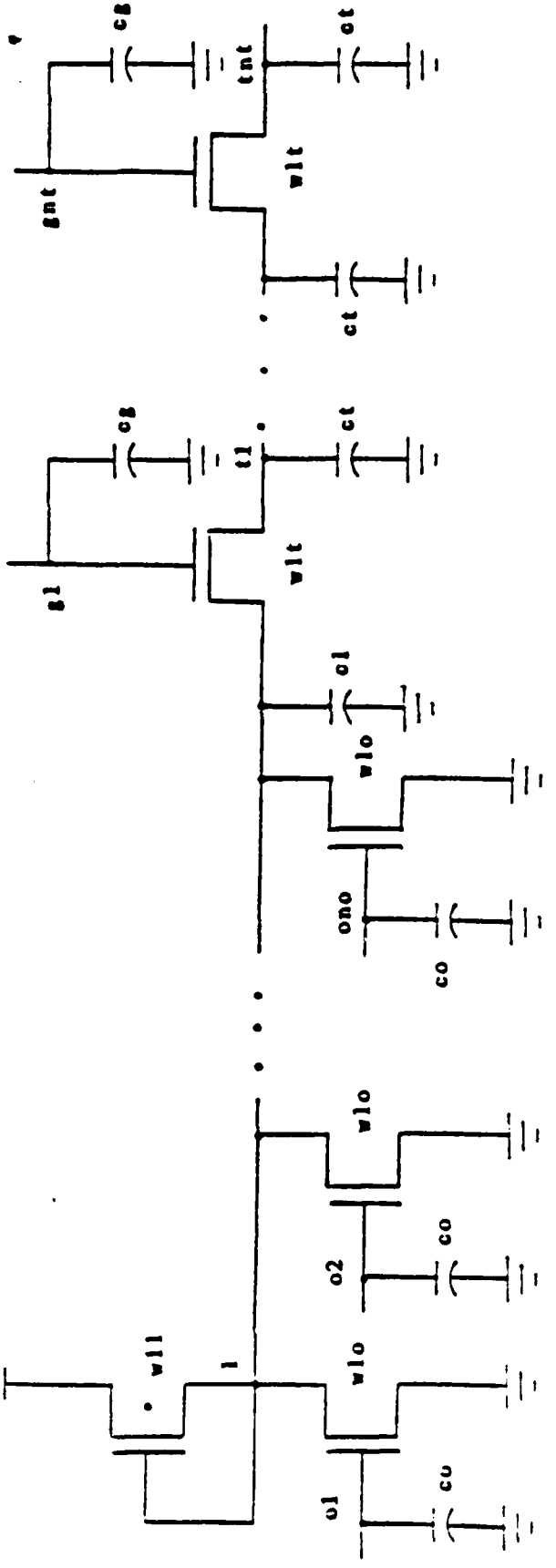


Fig. 7 Pass transistor network

An approach which is an extension of the Elmore [20] time constant approximation for an RC tree to the *pwl* case is given here.

The Elmore time constant is related to the impulse response of an RC tree. An RC tree is defined as an interconnection of resistors and capacitors with no loops. The resistors are restricted to be between nonground nodes only while the capacitors are only between nonground nodes and the ground. An example of an RC tree is depicted in Figure 8. The Elmore time constant or the delay is defined to be

$$T_e = \sum_i R_{ie} C_i$$

T_e is the time delay of node e, R_{ie} is the sum of the resistances common to the path between input and node i and to the path between input and node e and C_i is the capacitance of node i.

The Elmore time constant has been used to approximate rising and falling times of an RC tree, such as the one reported in [21]. The method in [21] does not work for the *pwl* approach, since it gives the upper and lower bound of the waveforms. Also, the function approximation is neither an exponential nor a straight line, so it is difficult to get the time when the transistor crosses to a new *pwl* interval. Consider, for example, a circuit consisting of an inverter and a pass transistor as shown in Figure 9a. Note that according to the above equation of the Elmore time-constant, the time-constant at node 1 depends on whether there is a path between node 1 and node 2. If that is the case, then the time constant used in solving for node 1 is

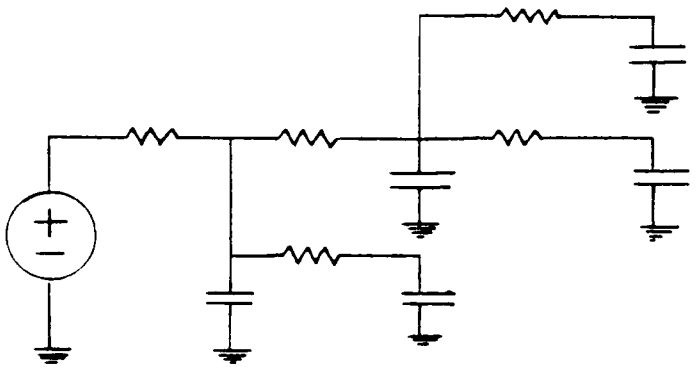


Fig. 8 An RC tree

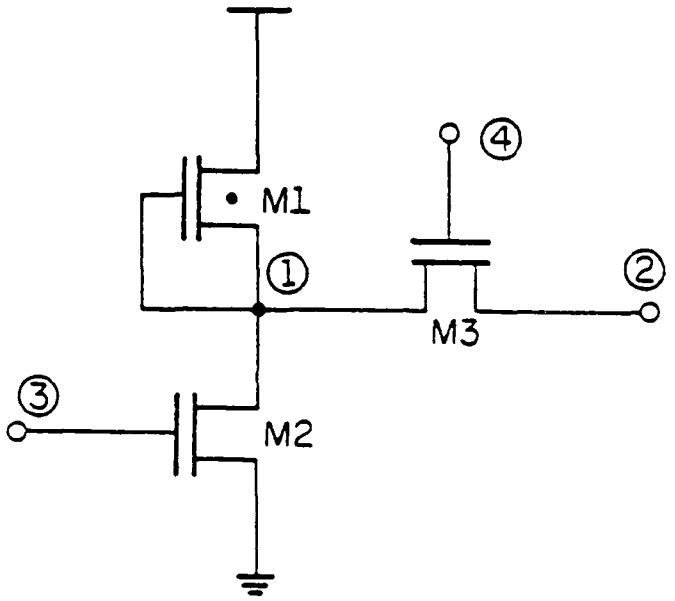


Fig. 9a A pass transistor circuit

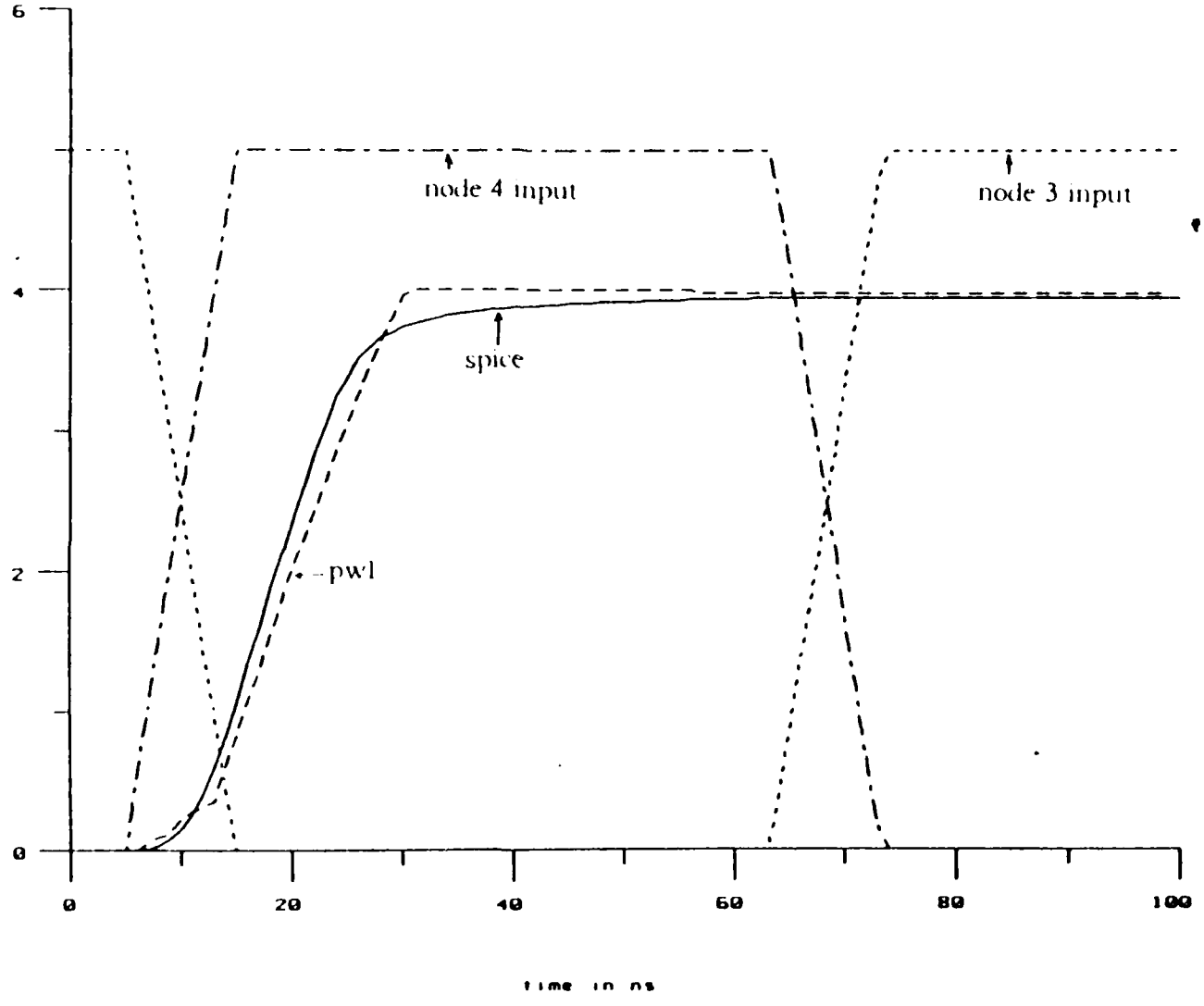


Fig. 9b Waveforms of the circuit in Fig. 9a

$$T_1 = R_{due\ inverter} * (C_1 + C_2)$$

Node 2 is more difficult to analyze. There are three cases to consider. If the pass transistor is off, there is no change to node 2 (Figure 10a). If there is a resistive path between node 1 and node 2 (Figure 10b), the time-constant used is

$$T_2 = R_{due\ inverter} * (C_1 + C_2) + R_{due\ pass\ transistor} * C_2$$

If the pass transistor is in the saturation region then the Elmore approach is not applicable because the equivalent resistor of the pass transistor is infinite (actually the pass transistor is represented by a controlled current source in series with a resistor, which is in parallel with a current source). Depending on whether node 2 is being charged up or being depleted, one of the following equations is used. In the former case (Figure 10c),

$$C_2 \frac{dv_2}{dt} = \text{current supplied to node 2}$$

$$C_2 \frac{dv_2}{dt} = (v_{gate} - v_2)/R_{pass} + i_{pass}$$

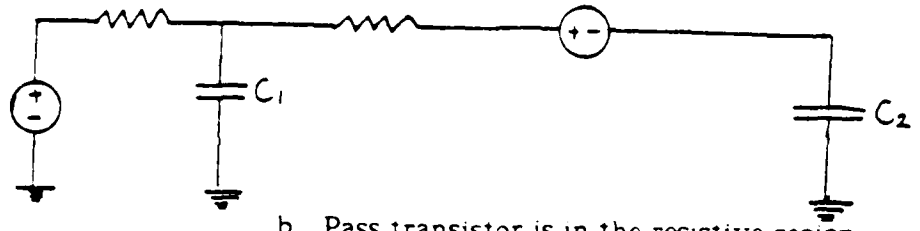
The solution to this equation is either a straight line or an exponential. If node 2 is being depleted (Figure 10d),

$$C_2 \frac{dv_2}{dt} = \text{current out of node 2}$$

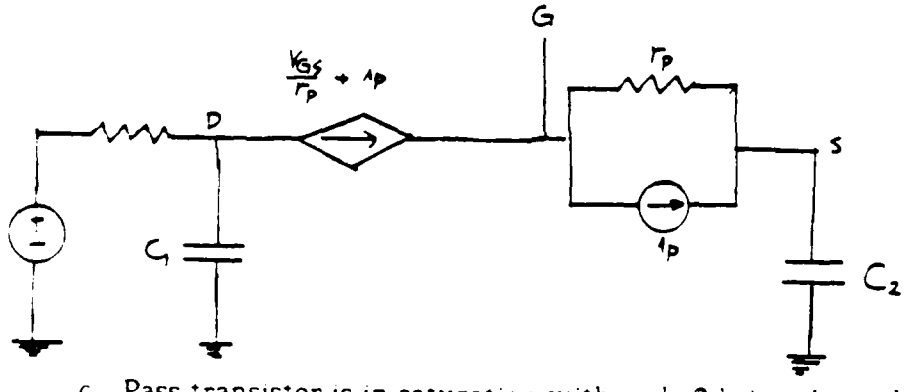
$$C_2 \frac{dv_2}{dt} = -(v_{gate} - v_1')/R_{pass} - i_{pass}$$

v_1' is held constant at its initial value until a new region is entered. The solution to this approximation is a straight line. The simulation result in Figure 9b of the circuit in Figure 9a indicates that the *pwl* result is reasonably close to the SPICE result.

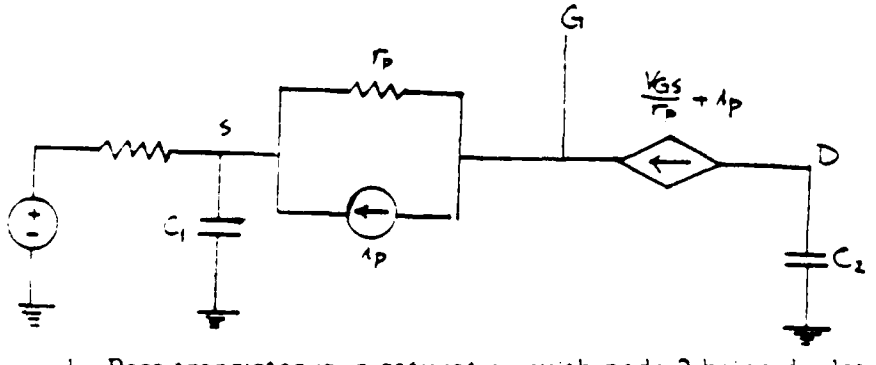
a Pass transistor is off



b Pass transistor is in the resistive region



c Pass transistor is in saturation with node 2 being charged up



d Pass transistor is in saturation with node 2 being depleted

Fig. 10 Piecewise linear model of a pass transistor

2.6.2. Circuits with internal nodes

The equations of a circuit with internal nodes (e.g., Nand gates) are of the form (assuming all capacitances are connected to the ground) $C_1\dot{x}_1 = a_{11}x_1 + a_{12}x_{i1} - a_{13}x_{i2} + \dots + a_{1n+1}x_{in} - w_1$

$C_{i1}\dot{x}_{i1} = a_{21}x_1 + a_{22}x_{i1} - a_{23}x_{i2} + \dots - a_{2n-1}x_{in} + w_2$

.

.

.

$C_{in}\dot{x}_{in} = a_{n1}x_1 + a_{n2}x_{i1} - a_{n3}x_{i2} + \dots + a_{nn-1}x_{in} - w_n$

where x_{ij} (j=1,2.....n) is an internal node variable and w_j is a constant.

One way of simplifying the above equations is to "lump" all the capacitances at the output and neglect all internal capacitances. In this case the equation becomes

$$(C_1 + \sum_{j=1}^{n} C_{ij})\dot{x}_1 = a_{11}x_1 + a_{12}x_{i1} + a_{13}x_{i2} + \dots - a_{1n+1}x_{in} - w_1$$

$$0 = a_{21}x_1 - a_{22}x_{i1} - a_{23}x_{i2} - \dots - a_{2n-1}x_{in} - w_2$$

.

.

.

$$0 = a_{n1}x_1 - a_{n2}x_{i1} - a_{n3}x_{i2} - \dots - a_{nn-1}x_{in} - w_n$$

Simulation of a nand gate is shown in Figure 11. From the simulation results we found that if the internal node capacitance is less then one tenth of the out-

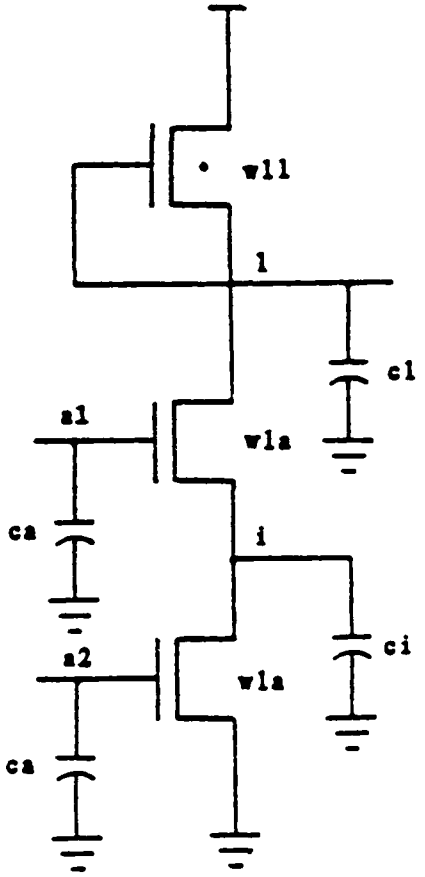


Fig. 11a Nand gate

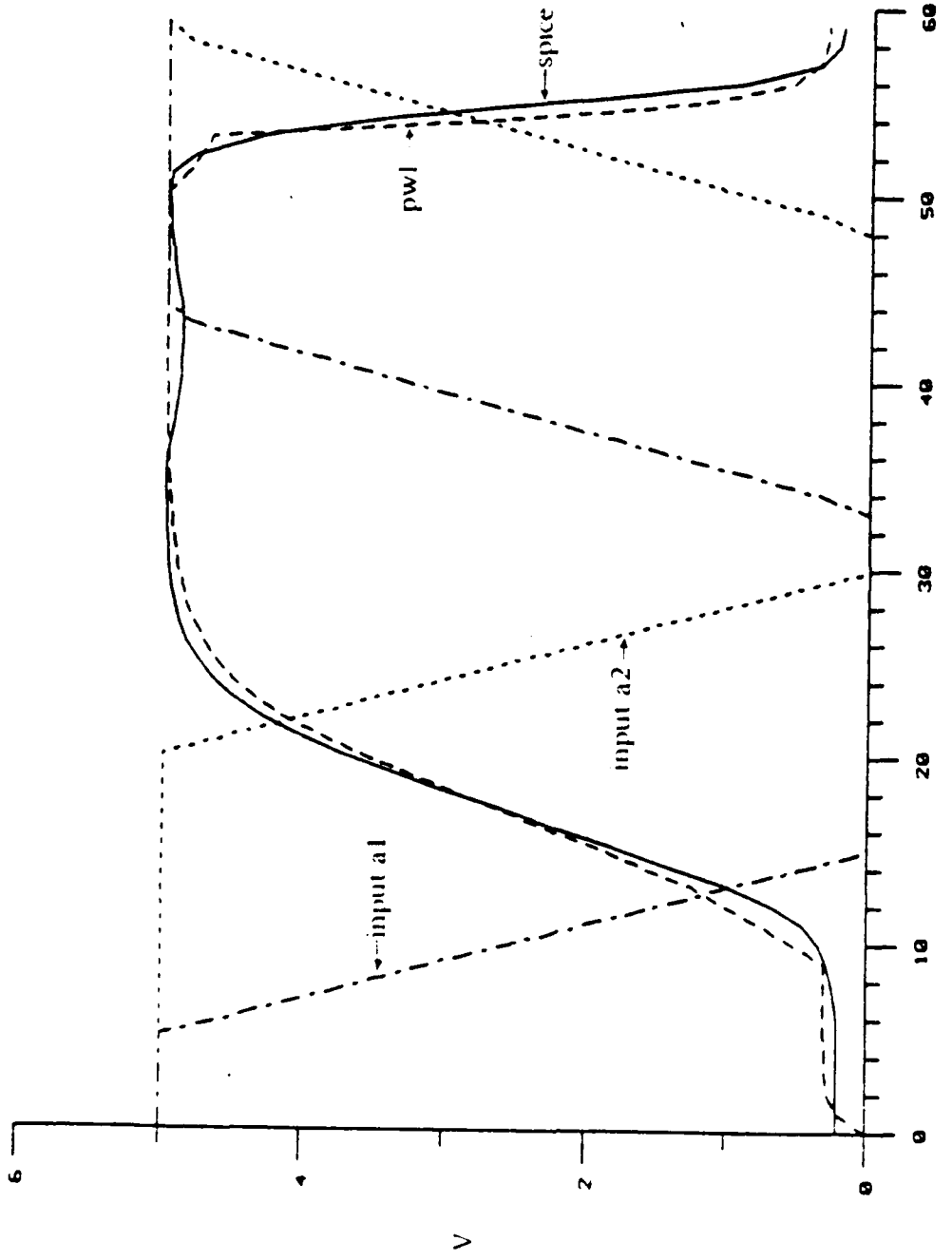


Fig. 11b Waveforms of the circuit on Fig. 11a

put capacitance our approximation gives reasonable results. If the internal node capacitances are too large then the direct method is employed.

2.6.3. Circuits with floating capacitors

For simplicity, we consider a two-dimensional problem of the form

$$C_{11}\dot{x}_1 + C_{12}\dot{x}_2 = a_{11m}x_1 + w_{1m} + y_1(t)$$

$$C_{12}\dot{x}_1 + C_{22}\dot{x}_2 = a_{21m}x_1 + w_{2m} + y_2(t)$$

The Elmore time-constant approximation is not applicable in this case since there is a capacitor between two nonground nodes, and in some cases resistors may be connected from the nodes to the ground. In this case, when applying the Gauss-Seidel method to this problem, the "staircase" approximation of the waveform does not work since the time derivative at the breakpoints is infinite. Therefore, a ramp approximation is made for falling and rising waveforms. The derivative of a ramp gives a constant function which is suitable for the approach described above. For a test circuit a bootstrap circuit as shown in Figure 12a is used. The waveforms are shown in Figure 12b. The ramp approximation is chosen as dv/dt at the midpoint of the rising or falling input. Four iterations are needed to get convergence.

The above approximation methods have their drawbacks. The method in which the Elmore time-constant approach is utilized gives good results when the pass transistor network is small. When the network consists of more than three pass transistors, the error of the pass transistor voltages becomes large. Therefore, for pass transistor networks the conventional approach, where the

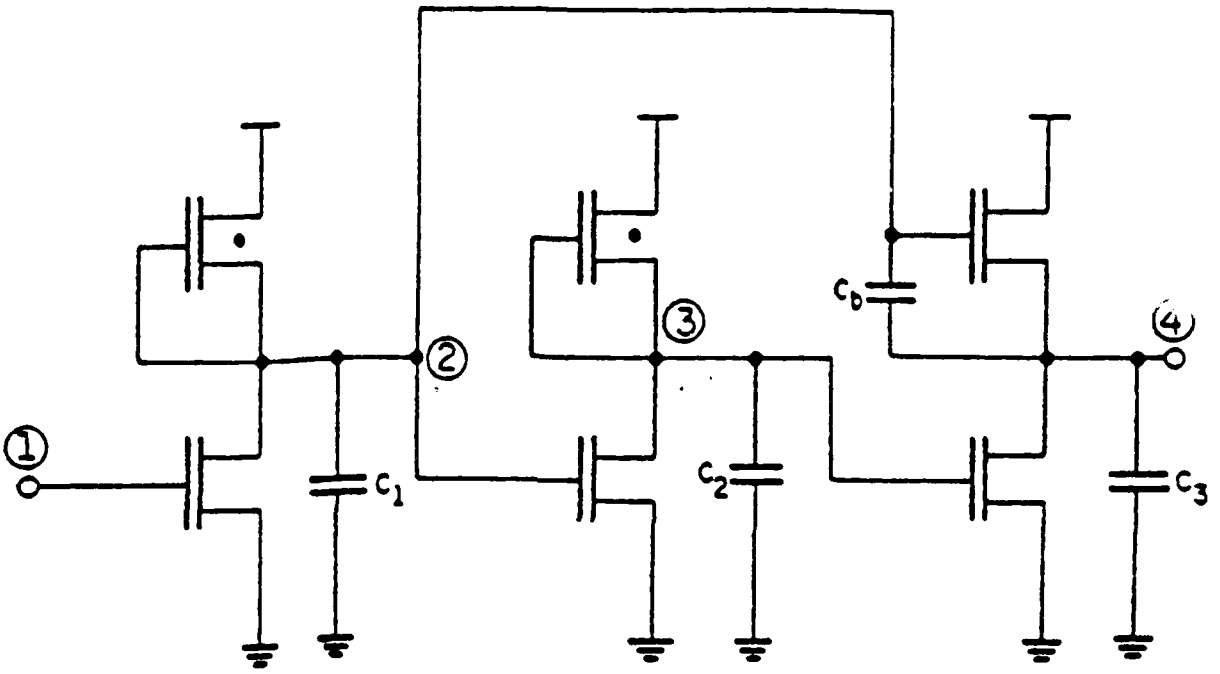


Fig. 12a Bootstrap circuit

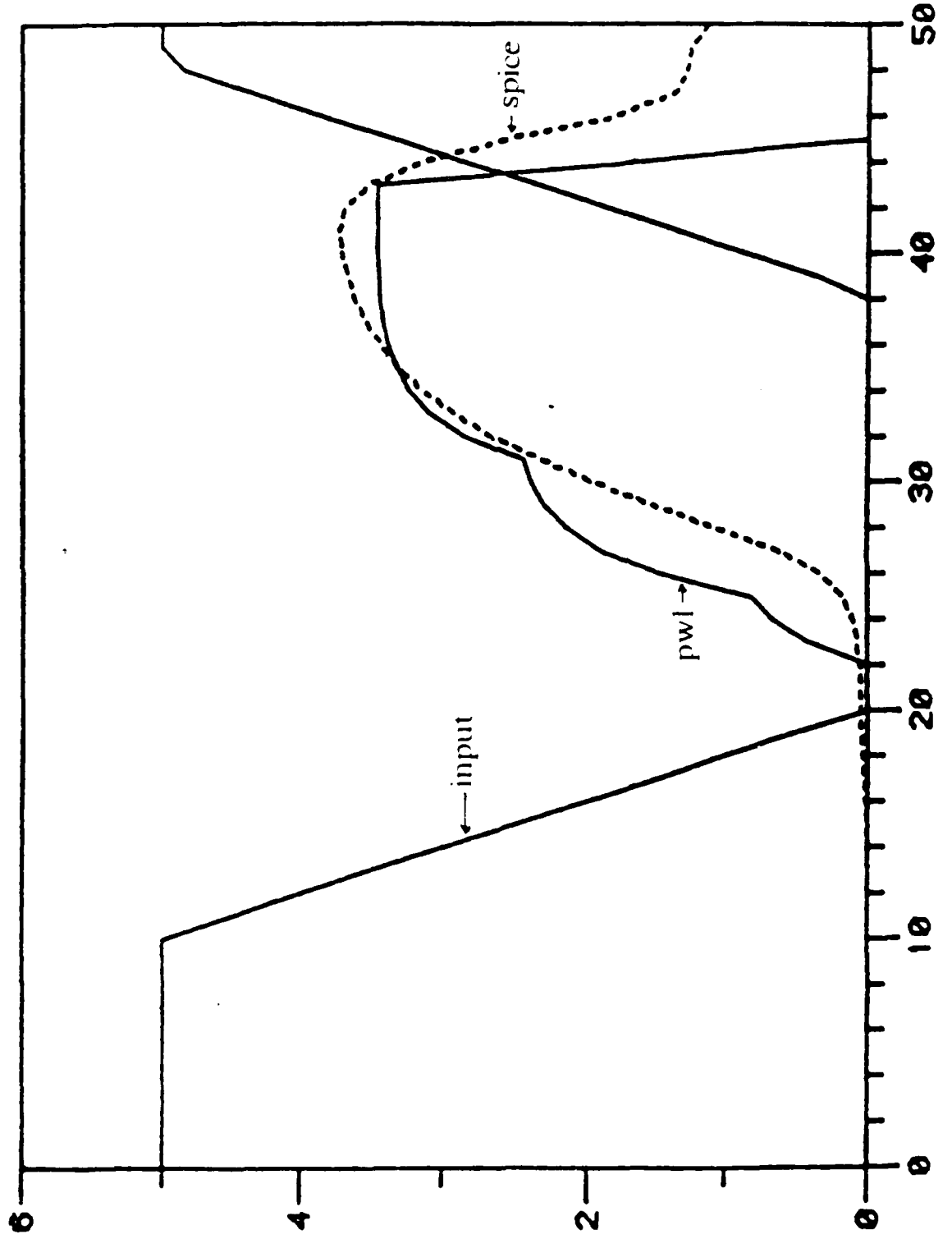


Fig. 12b waveforms of the circuit on Fig. 12a

time derivative is discretized using the Backward Euler Method and where analysis at the linear level is performed, is utilized. Similarly, when the internal capacitances become large compared to the output capacitance, the approximation method for nand-gate type circuits as described above becomes less accurate. To obtain more accuracy the subcircuit needs to be solved in the same way as it is done by standard circuit simulation; that is, the time derivative is discretized using Backward Euler Formula, and then the equations are solved at the linear level.

The method of the Gauss-Seidel waveform relaxation *pwl* is fast for analyzing simple gates such as inverter, nor, nand gates. The computation efficiency is due to the fact that there is no need to calculate the voltages at each time point. As long as the transistors remain in the same regions, the solution of the equation is either a straight line or an exponential. Another advantage is due to the fact that the solution is obtained using the waveform relaxation approach, which solves the equations at the differential equation level, and hence, there is no need of transforming the differential equation into the linear level. The drawback is that the method works well only for simple circuits such as the inverter, nor and nand gates, while for other types of circuits the methods can be very slow. In summary, the first method, the *pwl* analysis on simplices, which has good convergence and gives accurate waveforms, is very slow. The second *pwl* method, which solves the *pwl* circuit by inspection, is fast but limited in the type of circuits that can be accurately solved. Realizing the drawbacks of the methods described above, a *pwl* method which is quite fast but accurate is desirable. To obtain results as accurate as those from stan-

dard circuit simulators, an iterated relaxation-based *pwl* method is followed. To speed up convergence, and thus reduce computation time, a dynamic partitioning method is developed. Solving the *pwl* circuit equations with a new partitioning method, which is performed dynamically and efficiently, is described in Chapter 3.

CHAPTER 3

DYNAMIC PARTITIONING APPOACH FOR PIECEWISE LINEAR CIRCUITS

3.1. Introduction

The two *pwl* methods described in the previous chapter, namely the fast *pwl* and *pwl* on simplices methods, require that the strongly connected components in a circuit be solved either as a whole or using a relaxation method. Solving the strongly connected components as a whole might be too expensive because the blocks could be large. The relaxation method is preferable. However, where to break the loops of scc to start the relaxation method so that the number of iterations of the relaxation method is minimized is not known. The method followed is to cut the loop randomly and assign the corresponding node voltages to the previous values and start the relaxation process. Note that where the loop is cut is fixed throughout the simulation.

In this chapter we will describe a novel way of breaking the strongly connected components dynamically and naturally, so that the smaller partitioned subcircuits are manageable for analysis. Review of other methods are mentioned first.

3.2. Dynamic Partitioning

There has been an interest in partitioning large circuits into loosely coupled subcircuits. Specifically, in [33] the partition of MOS circuits is obtained by calculating the equivalent conductances and capacitances of two adjacent nodes. If the calculated values exceed some predetermined values then the two nodes are grouped together. This partition is done only once at the beginning of the simulation of the MOS circuits. A similar approach for bipolar circuits is described in [34], except here the partition is performed dynamically. The calculation/partition is not done at each iteration, since this would be too costly. Only when an iteration threshold is exceeded is a repartitioning performed. At this point it is expected that the speed up in computation is dominant over the repartitioning. Recently, a partitioning based on checking the coupling terms of the following nodal equation is proposed [18].

$$C_n \frac{dV_n}{dt} + G_n V_n - \sum_{j \in J} C_{nj} \frac{dV_j}{dt} - \sum_{j \in J} G_{nj} V_j = \sum_m I_m \qquad (3.1)$$

where V_n is the voltage at node n, V_j is the voltage at node j, J is the set of nodes connected to node n. C_n is the sum of capacitances connected to node n, C_{nj} is the sum of the capacitances connected between nodes n and j. G_n is the sum of conductances connected to node n and G_{nj} is the conductance between node n and node j. I_m is the current source connected to node n. If the coupling terms $\sum_{j \in J} C_{nj} \frac{dV_j}{dt}$ and $\sum_{j \in J} G_{nj} V_j$ are negligible compared to the right-hand side, then the coupling between node n and node $j \in J$ is negligible, and therefore the partition is performed between nodes n and j.

In our case the dynamic partitioning is applied to circuits consisting of transistors that have been piecewise linearized. The transistor model used is the Meyer's model [25]. The model is piecewise linearized; at each *pwl* region a particular type of transistor (load, driver or pass transistor) is represented by a conductance and a current source. These conductances and current sources values are stored in a table so that during transient analysis a table-lookup method can be employed. More details of the model are presented in section 2.2.3 and Appendix A. The partition relies on the comparison of integers indicating regions of piecewise linearized transistor operations, and it is done at each iteration of the Gauss-Jacobi or Gauss-Seidel method.

3.3. Piecewise Linear Dynamic Partitioning

Let the system of *pwl* algebraic-differential equations describing the circuit be written as

$$C_i \dot{\mathbf{x}}(t) = A_i \mathbf{x}(t) + \mathbf{b}_i + \mathbf{y}(t), \quad i=1,2,\ldots,r \tag{3.2}$$

where $C_i \in R^{n \times n}$ is the matrix representing piecewise linearized capacitors in the circuit, A_i and $\mathbf{b}_i$ represent piecewise linearized transistors and $\mathbf{y}(t)$ the input waveforms. The subscript denotes a particular region of the piecewise linearized elements.

Applying an implicit integration formula to (3.2), we get

$$C_i \frac{\mathbf{x}_{n-1} - \mathbf{x}_n}{h} = A_i \mathbf{x}_{n-1} + \mathbf{b}_i + \mathbf{y}(t_{n-1})$$

The Newton-Raphson iterative scheme is used to solve the nonlinear algebraic equations. Then at each time step, one solves

$$(C_i - hA_i)\,x_{n+1} = h(b_i + y(t_{n+1})) + C_i\,x_n$$

$$\left(\frac{C_i}{h} - A_i\right)x_{n+1} = b_i + y(t_{n+1}) + \frac{C_i}{h}\,x_n = s_i \qquad (3.3)$$

Consider first the case where C_i is *diagonal* (no floating capacitors) with capacitors from nodes to ground. Equation (3.3) can then be written as

$$[\hat{A}_i]\,x_{n+1} = s_i$$

where the off-diagonal elements of $\hat{A}_i$ are created by the resistive part of the circuit. The aim of our dynamic partitioning approach is to order the circuit variables so that the matrix $\hat{A}_i$ is block diagonal, with each diagonal block being as lower triangular as possible. At every iteration point the values of the off-diagonal elements of $\hat{A}_i$, and consequently the structure of $\hat{A}_i$, are determined by the local and global connectivity of the nodes in the circuit.

The local connectivity of the nodes is then determined by the slopes of the characteristics of the resistive elements at the iteration point. In the *pwl* case, these values depend on the region combination of the characteristics equations, which in our case are the MOS transistor characteristics. Let us consider the transistor as a three-terminal device, as shown in Figure 4. The contribution of a given transistor to the circuit matrix is as follows :

$$\begin{vmatrix} +g_D & -g_S & +g_S-g_D \\ -g_D & +g_S & -g_S+g_D \\ 0 & 0 & 0 \end{vmatrix} \qquad (3.4)$$

Note that since the gate-to-drain and the gate-to-source capacitances are ignored, the contribution to the row corresponding to the gate G is zero. Since there are three regions in each of the two two-terminal *pwl* branches in the transistor model as described in Chapter 2 above, there are nine possible region

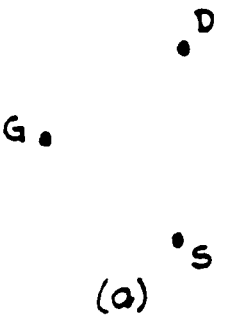
combinations in which the transistor operates. The values of g_D, g_S and $g_D - g_S$ in each of the regions are listed below.

Table I : Transistor connectivity.

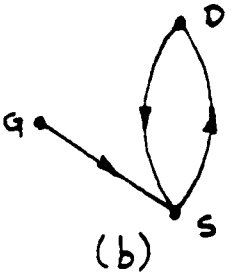
REGION	g_D	g_S	$g_D - g_S$	connectivity
(1,1)	0	0	0	Figure 13a
(1,2)	0	g_{S_2}	$-g_{S_2}$	Figure 13b
(1,3)	0	g_{S_3}	$-g_{S_3}$	"
(2,1)	g_{D_2}	0	g_{D_2}	Figure 13c
(3,1)	g_{D_3}	0	g_{D_3}	"
(2,2)	g_{D_2}	g_{S_2}	0	Figure 13d
(3,3)	g_{D_3}	g_{S_3}	0	"
(2,3)	g_{D_2}	g_{S_3}	$g_{D_2} - g_{S_3}$	Figure 13e
(3,2)	g_{D_3}	g_{S_2}	$g_{D_3} - g_{S_2}$	"

where the value of g_{D_2} is equal to g_{S_2} and the value of g_{D_3} is equal to g_{S_3}. Note that there are only two regions, namely, regions (2,3) and (3,2), in which the entries in rows D and S in (3.4) are all nonzero. The local transistor connectivity is then determined by checking the region as shown in Table I and Figure 13. Note that if the drain or the source is connected to the ground, only one row in (3.4) needs to be considered. Consequently, the connectivity of the circuit depends on the operating regions of the transistors. Hence, the structure or the zero-nonzero pattern of $\hat{A}_i$ can be determined from the transistor regions without any computation. This fact leads to the feasibility of performing efficient dynamic partitioning. The local connectivity in turn affects the global

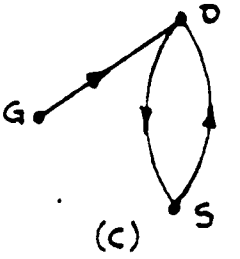
Region (1,1)



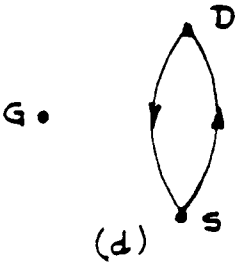
Regions (1,2) & (1,3)



Regions (2,1) & (3,1)



Regions (2,2) & (3,3)



Regions (2,3) & (3,2)

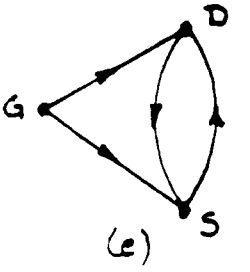


Fig. 13 Transistor connectivity

connectivity; that is, the local interconnection of drain, gate and source defines the overall interconnection of nodes in the circuit. The global connectivity of the nodes is then determined by applying a depth-first-search technique [14].

Because of the nature of digital MOS circuits, the above partitioning produces a block-diagonal circuit matrix with most of the blocks in lower triangular form, even for sequential circuits. The partitioning, of course, varies with the iteration points. We assume that there is a capacitance from every node to ground; therefore, for finite time step h the diagonal blocks are nonsingular. Thus at each iteration, the linear system in (3.4) in most cases is solved in one sweep using forward substitution, with the possibility of the diagonal blocks being solved in parallel.

When floating capacitors are allowed in the circuit, then the matrix cannot be in Block Diagonal or Block Lower Triangular form anymore; hence, one-sweep iteration is impossible. A typical matrix at one iteration when floating capacitors exist in the circuit is shown in Figure 14. In this case a combination of dynamic partition and Gauss-Seidel type of iteration is employed. The dynamic partition is applied to the transistors in the circuit, assuming the small-valued floating capacitors, such as the gate-source and the gate-drain capacitors, do not exist. A large-valued floating-capacitor such as a bootstrap capacitor is assumed to establish a connection between the nodes where it is connected. From experiments on some circuits the threshold value is the sum of the grounded capacitors which the floating capacitor is connected to. The floating capacitors that are not included in the partitioning will create feed back and feedforward terms within and between the diagonal blocks created by the

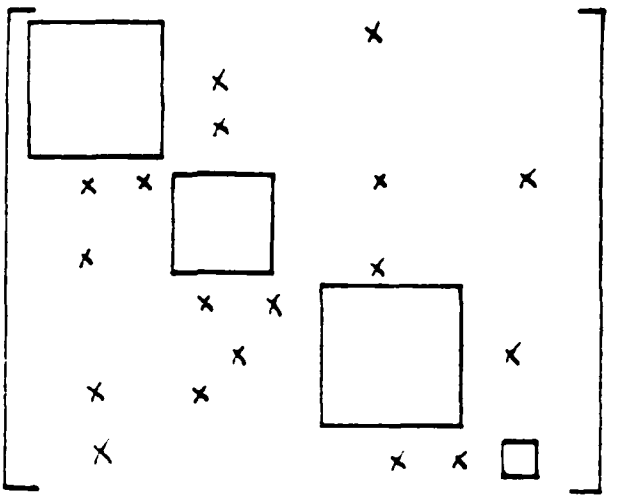


Fig. 14 A typical matrix when floating capacitors exist in the circuit

partitioning. In this case, Gauss-Seidel iteration is used in solving (3.4).

When a full-blown nonlinear transistor model is used, the method of checking the local connectivity needs to be generalized. Instead of region comparison, voltage comparison is performed on each transistor. For example, from Table I, in *pwl* case one concludes that the gate is independent of the source-drain part when the gate-drain region is equal to the gate-source region. In the general case the gate is independent from the source-drain part when the difference between the gate-source voltage and the gate-drain voltage is within some tolerance Δ V. Physically, it means that the source-drain current is independent of the gate voltage when the gate-source voltage is close to the gate-drain voltage.

As an example consider a simple 5-stage-ring-oscillator shown in Figure 15. A worst-case partitioning approach would treat all nodes as one block. In our case this one block is partitioned into smaller subblocks. A Newton-Raphson, Gauss-Seidel method is used to solve the circuit. Let us consider a piecewise linearized driver transistor with breakpoints 0, 1.5, 2.75 and 5 (Figure 3) and similarly a load transistor with breakpoints -5, -1.75, -1 and 0. Assume that a falling step input is applied and dc values for the nodes have been calculated [(node,voltage) : (2,0), (3,5), (4,0), (5,5), (6,0)]. By checking the table of the driver and comparing the regions of the transistor operation, one concludes that at this initial state all the nodes are decoupled from one another. At other iterations the partition changes. For instance, at the 9 nanoseconds the voltages of the nodes are [(node,voltage) : (2,2.885), (3,2.815), (4,0.15), (5,5), (6,0.15)]. Checking the driver table one obtains the regions of

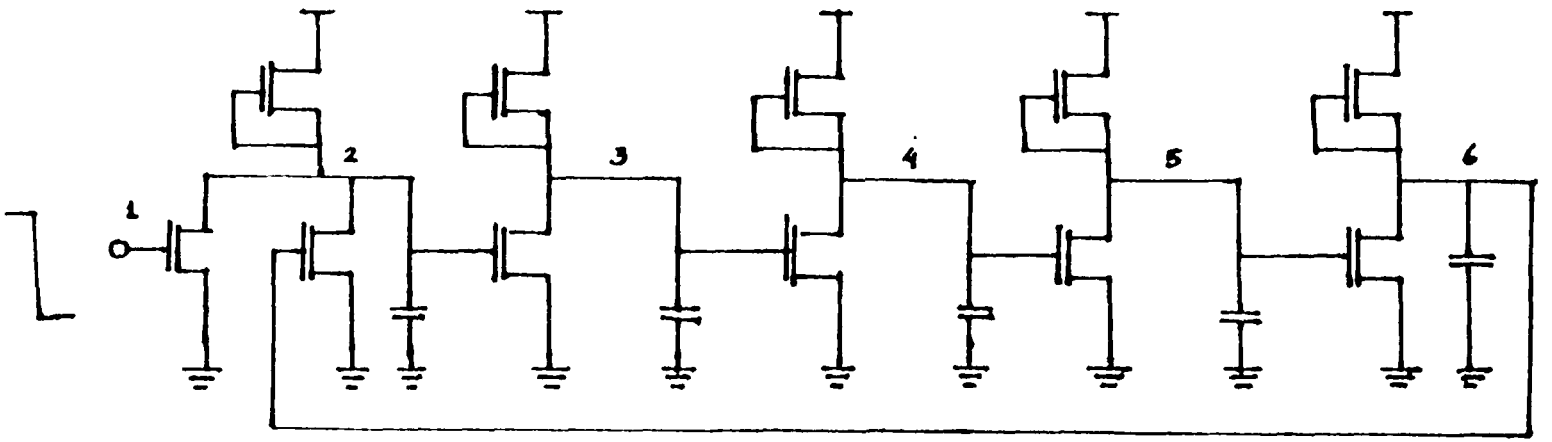


Fig. 15 Five-stage ring oscillator circuit

the driver of the first inverter to be (1,1). The regions of the driver of the second inverter are (2,3). This indicates that in the matrix the drain of this driver depends on its gate. Similarly, the regions of the driver of the third inverter are (3,2) and hence the drain node depends on the gate node. As a result, the nodes 2, 3 and 4 are in one subblock. By applying the same procedure one finds that node 5 and node 6 are in two separate subblocks. Figure 16 shows the partitions at three instances. Figure 16a shows that the nodes 2,3 and 4 are in one block which is lower triangular; node 5 is in one block and node 6 is in another. All blocks are completely decoupled. Solving the matrix using the Gauss-Seidel iterative method dynamic partitioning and worst-case partitioning have the same effect, in that the voltages are obtained in one sweep of calculation. Figure 16b shows another partition using the dynamic partitioning method at a different instance while Figure 16c shows a partition using the worst-case one at the same instance as Figure 16b. Using the dynamic partitioning method (Figure 16b) the solution is obtained in one sweep. Nodes 6 and 2 are solved together as one block while the rest of the nodes are in separate blocks. On the other hands, using the worst-case partitioning approach (Figure 16c) would require more than one iteration due to the existence of the upper-diagonal element. Figures 17a and 17b show the corresponding graph representation of Figures 16a and 16b, respectively.

To reduce the computation time even further, the nonactive partitioned subcircuits could be identified. The nonactive (latent) subcircuits do not need analysis. The active subcircuits consist of transistors that do not change regions but their terminal nodes are active. An active node of a circuit is the one that

---

(node,initial voltage) (2,2.885) (3,2.815) (4,0.15) (5,5.0) (6,0.15)

time 0.9000d-08 nodes (2,3,4,5,6)

0.130d−03	0	0	0	0
0.250d−03	0.130d−03	0	0	0
0	0.150d−03	0.230d−03	0	0
0	0	0	0.160d−03	0
0	0	0	0	0.380d−03

node voltages 0.2885d-01 0.2626d+01 0.3287d+00 0.5000d+01 0.1500d+00

Fig. 16a Matrix of the ring oscillator at 9ns

(node,initial voltage) (2,4.867) (3,0.15) (4,4.125) (5,0.1684) (6,1.663)

time 0.2300d-07 nodes (6,2,3,4,5)

0.130d−03	0	0	0	0
0.100d−03	0.160d−03	0	0	0
0	0	0.380d−03	0	0
0	0	0	0.160d−03	0
0	0	0	0	0.380d−03

node voltages 0.1663d-01 0.4765d+01 0.1500d+00 0.4109d+01 0.1684d+00

Fig. 16b Matrix of the ring oscillator at 27 ns (dynamically partitioned)

0.160d−03	0	0	0	0.100d−03
0	0.380d−03	0	0	0
0	0	0.160d−03	0	0
0	0	0	0.380d−03	0
0	0	0	0	0.130d−03

Fig. 16c Matrix of the ring oscillator at 27 ns (worst-case partitioned)

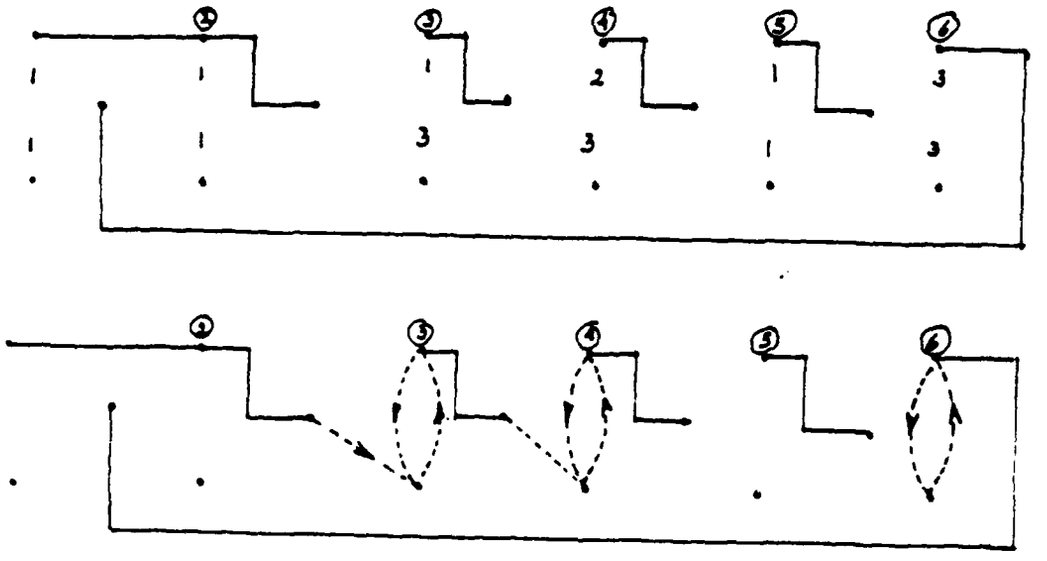


Fig. 17a Partition at 9ns

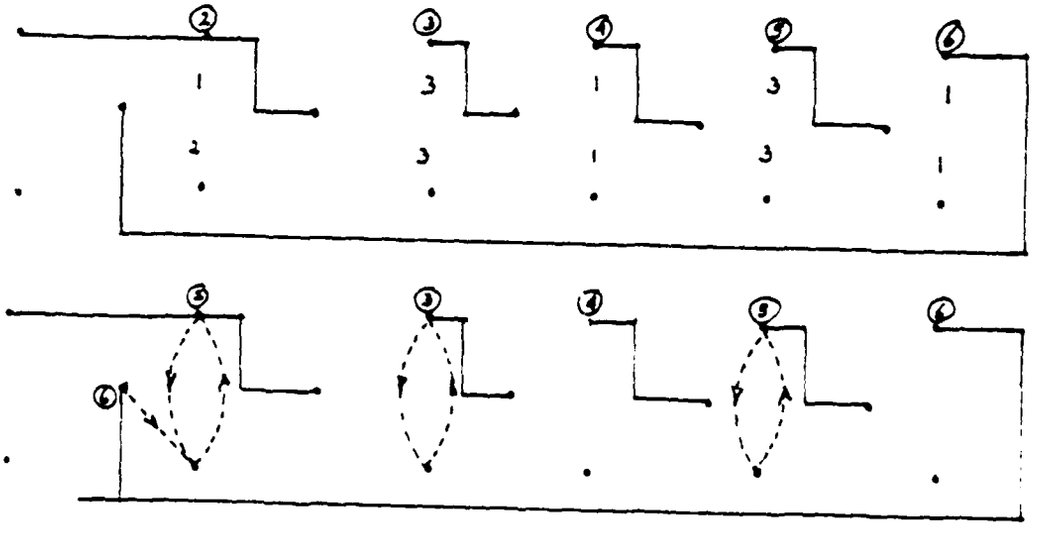


Fig. 17b Partition at 27ns

violates at least one of the following [2]:

(1) $V_m(t_n) - V_m(t_{n-1}) \leq \epsilon_a + \epsilon_r \max(V_m(t_n), V_m(t_{n-1}))$

m=1,2....

where ϵ_a and ϵ_r are the absolute and relative error tolerances for voltages.

(2) $I_m(t_n) - I_m(t_{n-1}) \leq \epsilon_c + \epsilon_r \max(I_m(t_n), I_m(t_{n-1}))$

m=1,2....

where ϵ_c is the absolute error tolerance for current.

(3) $h_{n-1} \dfrac{I_m(t_n) - I_m(t_{n-1})}{Q_m(t_n) - Q_m(t_{n-1})} > 1$

m=1,2....

where h_{n-1} is the time step taken by the program at t_{n-1} and Q_m is the charges of the capacitor at node m.

In timing analysis only the first and second rules are checked.

As an example let us consider the ring oscillator example. The following table shows how partitions change during the solution process. The numbers in the parenthesis show the node numbers that are in the same block, for example (2,3) indicates node 2 and node 3 are in the same subcircuit.

There are two important numerical processes that can be deduced from the table.

1. Repartitioning.

From time 0 to 5 ns the partition stays the same. At 5 ns, the partitioning changes, and stays the same until 9 ns when the partitions change again.

Table II : Partitions of the ring oscillator circuit

Time	Iteration	Partition
0ns	0	(2), (3), (4), (5), (6)
	1	(2), (3), (4), (5), (6)
5ns	0	(2), (3), (4), (5), (6)
	1	(2,3), (4), (5), (6)
	2	(2,3), (4), (5), (6)
9ns	0	(2,3), (4), (5), (6)
	1	(2,3,4), (5), (6)
	2	(2,3), (4), (5), (6)
	3	(2,3), (4), (5), (6)

Only those transistors that change regions are included in the repartitioning process.

2. Analysis.

All subcircuits that go through repartitioning must be analyzed. The subcircuits that are not repartitioned but whose node voltages change considerably must also be solved. The rest of the circuit that is not repartitioned and is latent need not be solved. For example, in the ring oscillator above, from time 0 to 5 ns, although the partitioning stays unchanged, nodes 2 and 3 start oscillating while nodes 4,5 and 6 remain latent. This means that only voltages of nodes 2 and 3 need to be solved. The ring oscillator, which is analog in nature, represents an extreme case: when time advances, all node voltages change. Digital circuits typically exhibit a greater degree of latency.

The above dynamic partitioning approach is performed on top of worst-case partitioning, which is performed once at the preprocessing step. The

worst-case partitioning is necessary to determine which parts of a circuit are large enough to require dynamic partitioning. Worst-case partitioning, which is also known as partitioning into dc-connected subcircuits, is based on worst-case transistor local connectivity as shown in Figure 13e.

CHAPTER 4

PARALLEL-VECTOR IMPLEMENTATION OF PIECEWISE LINEAR DYNAMIC PARTITIONING METHOD

In the last few years there has been a growing interest in developing CAD tools to run on parallel and on vector computers. The idea of parallel computation is that using N processors a program should run N times faster than if only one processor is used. In reality, the computing speed up is often smaller than the theoretical one. The idea of vector or pipeline computers is that by dividing a task into subtasks and by maintaining a flow of operand pairs in the analysis process the speed up can be increased.

To utilize the maximum capability of a vector and/or parallel computer, one needs to use the appropriate languages and algorithms. From the user's point of view, very little can be done about the language since usually it is given by the manufacturer who already tailors the language to the specific architectures of the machine. Given a particular machine architecture, one needs to design algorithms that can provide the best possible results.

The dynamic partitioning method described in Chapter 3 is well suited for implementation on a parallel machine with shared memory. The reason is that during the iterations exchanges of vertices and nodes among the blocks in the graph representing the circuit occur. In other words, there are exchanges of transistors among the partitioned subcircuits. Implementation of the method on a parallel machine with local memory would have a high cost of

intercommunication among the processors. The machine used in this study is Alliant FX/8. It a parallel-vector machine with 8 processors and a shared memory and, therefore, is suitable for implementation of the dynamic partitioning method.

The main iteration loop of the dynamic partitioning method consists of determining local connectivity, determining global connectivity and solving each partitioned subcircuit block. Figure 18 depicting the steps is shown on the next page. The box enclosed by broken lines will be explained later. This box later is modified to make the approach more efficient. Each process can be done in parallel as described in the following paragraphs. A general approach applied to each of the processes is described first.

In general, complete parallel vectorization is not feasible. Since vectorization of a loop prohibits subroutine calls, only parallelization is useful for most cases. The parallelization on the Alliant is performed by setting up a do loop, with a directive for concurrency. A typical format is as follow :

```
cvd concur
do 1 i=1,n
        call routine
1 continue
```

The loop contains a call to a routine that does a task. The concurrency is automatic; that is, no particular assignment of processors is necessary. Each of the available processors performs a subroutine call. When one processor finishes a job, it would automatically perform another call until all the n number of calls are completed. Each routine inside the concurrent do loop, in general, con-

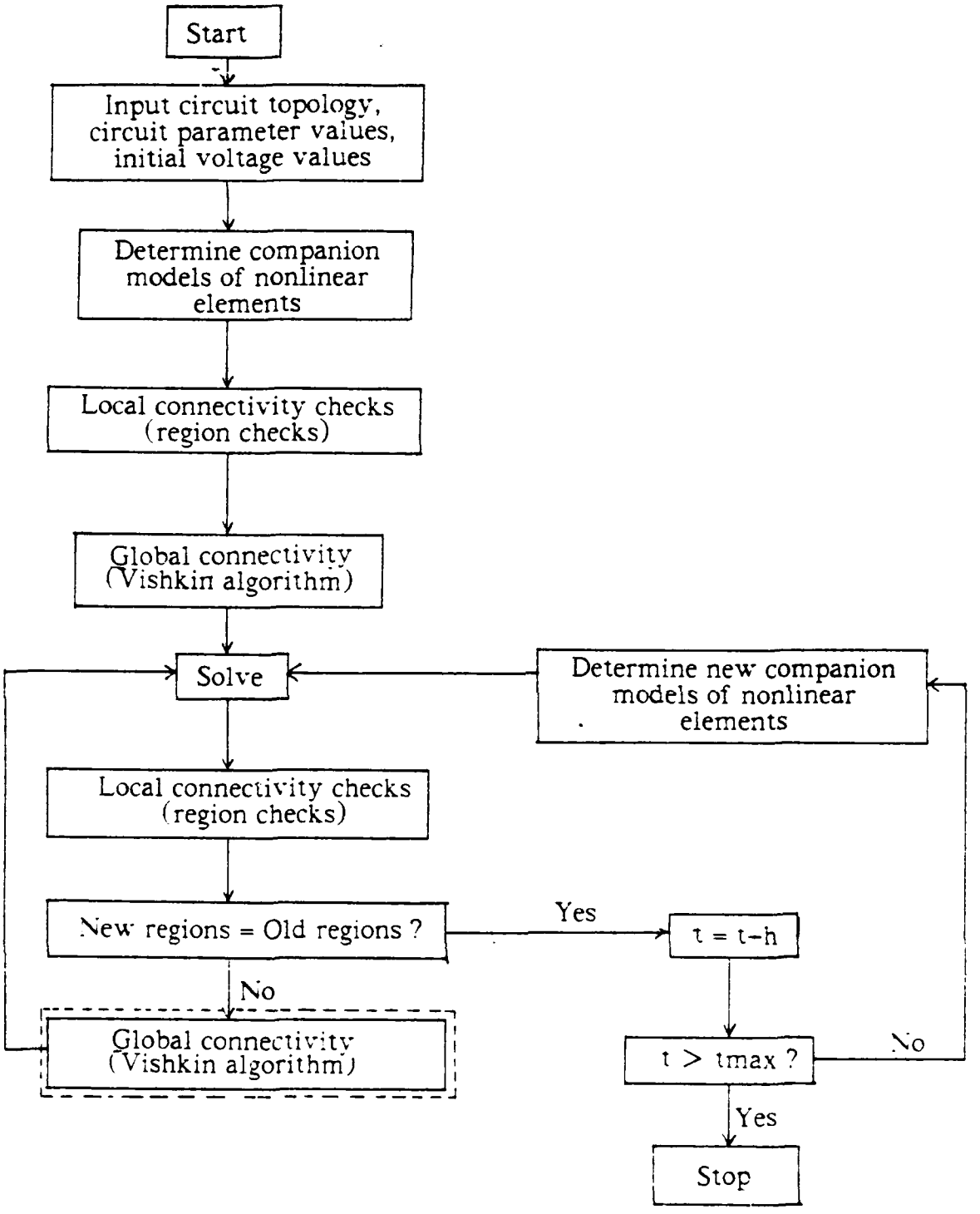


Fig. 18 Flowchart I : dynamic partitioning algorithm

tains a set of common blocks of global variables and a set of local variables. When concurrency is invoked, a stack is created so that each local variable has n different copies where n is the number of processors (at present n=8 on the Alliant). Each of the global variables is potentially accessible by many processors at the same time. This is not desirable because incorrect values would be stored. A lock is applied during the execution of the code that updates the global variable to prevent the concurrent execution by multiple processes. A special feature that the lock must have is that one instruction must check if a variable is free and if it is to set (or lock) the variable. This is important since if the setting is not done instantly, another processor might consider the variable to be free and attempt to set it.

The determination of local connectivity, global connectivity and solution of the variables follow the pattern described in the above paragraph.

For local connectivity the loop is

```
cvd concur
do 2 i=1,number of transistors
      call mostbl
2 continue
```

The input parameters to mostbl are the voltages of source, gate and drain, and the outputs are the regions of gate-drain and gate-source and the associated conductances and current sources. In this routine the connectivity of each transistor is determined and the corresponding edges between the source, drain and gate nodes are created (if applicable). These edges are needed for global connectivity determination.

The global connectivity is determined after transistor connectivity is completed. Before explaining the implementation details, parallel algorithms to determine the connectivity of a graph problem are described next. Hirschberg etal. proposed a method that solved the connected component problem of an undirected graph in $O(\log^2 n)$ time using $n^2/\log n$ processors [52], where n is the number of nodes in the graph. A variation of the method which requires an even smaller number of processors of $\max(n, e)$ is given in [53], where e is the number of edges in the graph. Another algorithm that determines the connected component of an undirected graph and uses an approach that is different from the ones above is proposed by Shiloac and Vishkin [54]. This algorithm determines the connected components in O(log n), but it requires 2e+n processors. Since the number of processors in a parallel computer is bound to increase in the future, this algorithm which requires more processors but determines the connected components in shorter time is chosen for our work. Another advantage is that the amount of temporary working memory in this case - O(log n) - is much smaller than for the one proposed in [52], which is of $O(n^2)$. Such a memory space requirement can be prohibitive when the size of the circuit is large.

The input to the algorithm of Shiloac and Vishkin consists of

- the vertices represented by the numbers 1,....n
- the edges specified by a vector e of length 2e in which edge (i,j) appears as a pair of directed edges <i,j> and <j,i>.

The output is a vector D [1:n] where D [i] points to the root node to which [illegible] is connected.

A temporary memory Q of length n is needed. The two main operations of the algorithm are

(a). Shortcutting : decreasing the height of a tree

(b). Hooking : reducing the number of trees.

An informal description of the algorithm is given first, followed by the more formal one. The notation $D_s(i)$ = j means that vertex i points to vertex j after the s^{th} iteration. Initially, each vertex points to itself, that is, $D_0(i)$ = i for i=1,...,n.

Informal description of the algorithm :

Step 1 : First shortcutting $D_s(i) \leftarrow D_{s-1}(D_{s-1}(i))$:

If, in $s-1^{th}$ iteration, node i points to some node j and node j points to another root k, then after the s^{th} iteration, point node i to node k (shortcutting).

Step 2 : Hooking trees onto smaller vertices of other trees. For all vertices that point to a root at the end of the previous iteration check if their neighboring vertices point to smaller vertices. If such a neighboring vertex j exists for a particular vertex i, then hook the tree to which i belongs onto $D_s(j)$.

Definition D: A tree is stagnant in the s^{th} iteration if it has not been changed in the first two steps of this iteration; that is, it has not been subjected to any shortcut operation, no tree has been hooked onto it, and it has not been hooked onto any other tree. A root of a stagnant tree is a stagnant root.

Step 3 : Hooking stagnant trees:

For all processors of vertices that point to a stagnant root, check if their neighbors point to a vertex of another tree. If such a vertex j is found, hook its tree onto $D_s(j)$.

Step 4 : Second shortcutting $D_s(i) \leftarrow D_{s-1}(D_{s-1}(i))$:

Same as step 1

A graphical procedure is given next.

Initially, each node points to itself.

After the hooking operation (on the edges), the nodes start to point to their neighbors with smaller node numbers.

Then after the shortcutting operation (on the nodes), the nodes point to other nodes further down.

The hooking and shortcutting operations are repeated until finally all the nodes point to the root.

A more complete description of the algorithm and the necessary arrays used are given next. The algorithm contains the vector Q which satisfies :

$Q(i) = s$ if after the second step of the s iteration there exists at least one vertex j pointing to i that does not point to i after the (s-1)th iteration.

$Q(i) < s$ otherwise.

Step 0 : Initialization. $D_0(i) \leftarrow i$, $Q(i) \leftarrow 0$, $s \leftarrow 1$, $s' \leftarrow 1$

In the following steps, $i \leqslant n$ indicates that the processors are working on the nodes, and $i > n$ indicates that the processors are accessing the pairs of edges (i_1, i_2).

While s'=s do

Step 1 : If $i \leqslant n$

then $D_s(i) \leftarrow D_{s-1}(D_{s-1}(i))$

if $D_s(i) \neq D_{s-1}(i)$

then $Q(D_s(i)) \leftarrow s$

Step 2 : If $i > n$

then if $D_s(i_1) = D_{s-1}(i_1)$

then if $D_s(i_2) < D_s(i_1)$

then $D_s(D_s(i_1)) \leftarrow D_s(i_2)$

$Q(D_s(i_2)) \leftarrow s$

Comment : If $D_s(i_1)$ has not been changed in Step 1, that is, it has pointed to a root, then the processor checks if i_2 is pointing to a smaller vertex. If that is the case, then it hooks the root which is $D_s(i_1)$ *onto* $D_s(i_2)$. Simultaneously, all the processors for which $D_s(j) = D_s(i_1)$ and $D_s(k) < D_s(i_1)$ try to update $D_s(D_s(i_1))$.

Step 3 : If $i > n$

then if $D_s(i_1) = D_s(D_s(i_1))$ and $Q(D_s(i_1)) < s$

then if $D_s(i_1) \neq D_s(i_2)$

then $D_s(D_s(i_1)) \leftarrow D_s(i_2)$

Comment : The processor checks if $D_s(i_1)$ is a root. If so, it checks by using Q if it is a stagnant root and if it is so it tries to hook it onto another tree. This is tried simultaneously by the processors such that $D_s(j) = D_s(i_1) \neq D_s(k)$.

Step 4 : If i $\leqslant$ n

then $D_s(i) \leftarrow D_s(D_s(i))$

Step 5 : If i $\leqslant$ n and Q(i) = s

then s' $\leftarrow$ s'+1

s $\leftarrow$ s+1

end while

Comment : As soon all the trees are stagnant, Q(i) < s for all i, $1 \leqslant i \leqslant n$, and thus s' will not be incremented while s is incremented, and the algorithm terminates.

In steps 1.4 and 5 the concurrency is across the nodes. In steps 3 and 4 the concurrency is applied to the edges.

During iterations the nodes in the graph remain the same while the edges change. The result is a graph that is repartitioned into blocks where each block is solved using one processor. A block consists of a root node and its corresponding leaves. The number of leaves varies from none (only 1 node in the block) to n where n is the number of nodes in the circuit. In the program the loop for solving the blocks is

cvd concur

do 3 i=1,number of blocks

call solve

3 continue

The output of the solve routine is the node voltages.

The next step is to concurrently do table-lookup for the transistors with the new voltages. If the resulting new gate-drain and gate-source regions are the same as the old ones, and if the new node voltages are within a tolerance of the old node voltages, then the solution is found. Time is then incremented by an automatically determined time step. Otherwise, the iteration (local connectivity, global connectivity, and solve) is repeated until convergence is obtained. Note that repartitioning is only done during the dc solution phase.

During transient analysis the circuit could be repartitioned a large number of times since repartitioning is potentially carried out at each iteration. It is then desirable to reduce the cost of repartitioning as much as possible. The algorithm described in the previous paragraphs repartitions the entire circuit. Since only a small part of the circuit experiences region changes, and therefore edge changes, the repartitioning needs to be performed only on this changing part. A modification of the original Shiloac and Vishkin algorithm which only repartitions part of the circuit is described next.

Modified Shiloac and Vishkin algorithm :

Once the Shiloac and Vishkin algorithm is applied to the entire circuit, the repartitioning is performed on selective parts of the circuit as follows :

Step 1. The gate-drain and gate-source edges of a transistor of the $n+1^{th}$ iteration are compared to the ones of the n^{th} iteration. If

different, then check if the root of the source has been flagged. The flag indicates if the root has been added to the list of nodes that need repartitioning. If the root node has not been flagged, flag it and add the root node to the list. The same checking is performed on the root of the drain (and the gate if necessary).

Step 2. Each root node in the list has pointers to the list of transistors.

These transistors have their source and drain nodes as the leaves of the root node. The edges from these transistors represent edges (that is ordered pair <i,j> where i and j are the nodes connected to the edges) in the Shiloac and Vishkin algorithm. The original algorithm considers *all* the edges in the graph to be partitioned. The nodes of the transistors are the vertices in the algorithm. Again, in the original algorithm, *all nodes in the graph are considered.*

An example showing the method is given in Figure 19, where the circles are the nodes, a circle enclosing a star is a root and the solid lines are the edges of the graph. The edges are created during local connectivity checks of the transistors. The broken lines with dots are also edges; however, these are either new edges created or old edges removed on the n^{th} iteration. The directed broken lines are pointers created during the Shiloac and Vishkin algorithm. On the top figure (n^{th} iteration) there are three subgraphs with three roots. During the n^{th} iteration one edge is deleted and one edge is created. These edge changes affect only two of the subgraphs. Therefore, the repartition is performed only on these collections of edges and nodes. The third subgraph is not affected by edge changes

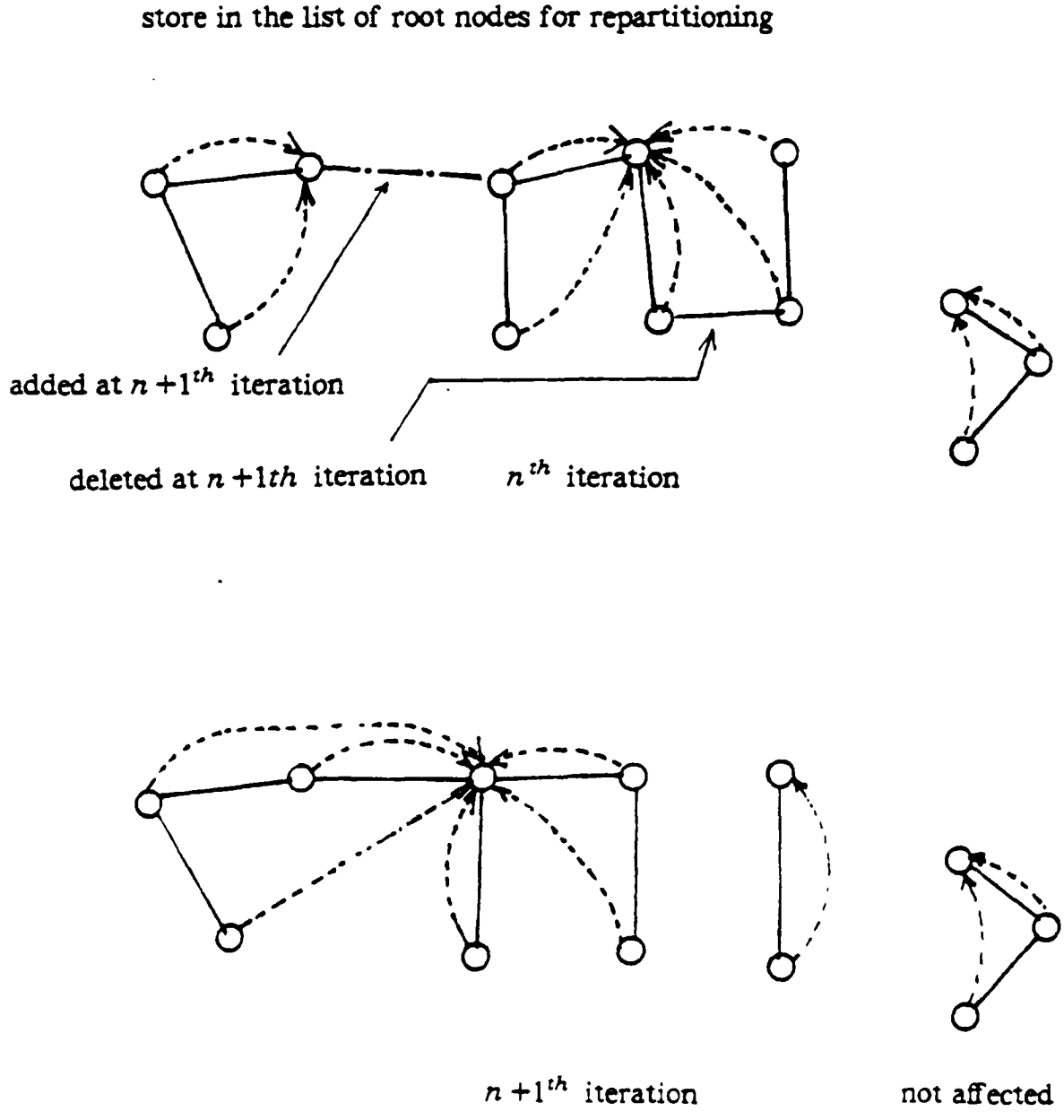


Fig. 19 Repartitioning of only some parts of the circuit

so there is no need to repartition this part. The resulting repartitioned graph at $n+1^{th}$ iteration is shown at the bottom of the Figure 19.

Besides repartitioning only the necessary parts, the computation time can be reduced even further by analyzing only the active subcircuits. Selecting the active subcircuits is explained in the preceding chapter. An example showing parts that need repartitioning (and analysis) and those that do not need repartitioning but require analysis is shown in Figure 20.

The symbols of solid lines, broken lines, broken lines with dots, circles and circles enclosing stars in Figure 20 have exactly the same meaning as the ones in Figure 19. At the top of Figure 20 is the graph at n^{th} iteration. There are four subgraphs with four roots. Note that there is one edge being formed at n^{th} iteration. The two subgraphs affected by the new edge are repartitioned, while the other two subgraphs do not have any edges deleted or created; hence, no repartition is necessary on these subgraphs. However, one of these two subgraphs contains active transistors. This particular subgraphs is solved (no repartitioning) and the other subgraph is neither repartitioned nor solved. The new graph at $n+1^{th}$ iteration is shown at the bottom of Figure 20.

Flowchart II, showing the modifications, is shown on page 85. The modification is done to the box enclosed by broken lines on Flowchart I, which is shown in page 73. Filter I separates the roots at the end of i^{th} iteration into two groups, one containing roots affected by changes of edges. This is the group that is being repartitioned using the Vishkin and Shiloac algrorithm and is later solved. The rest of the roots are partitioned even further into two groups, one containing roots with some active transistors. This group is later solved. The

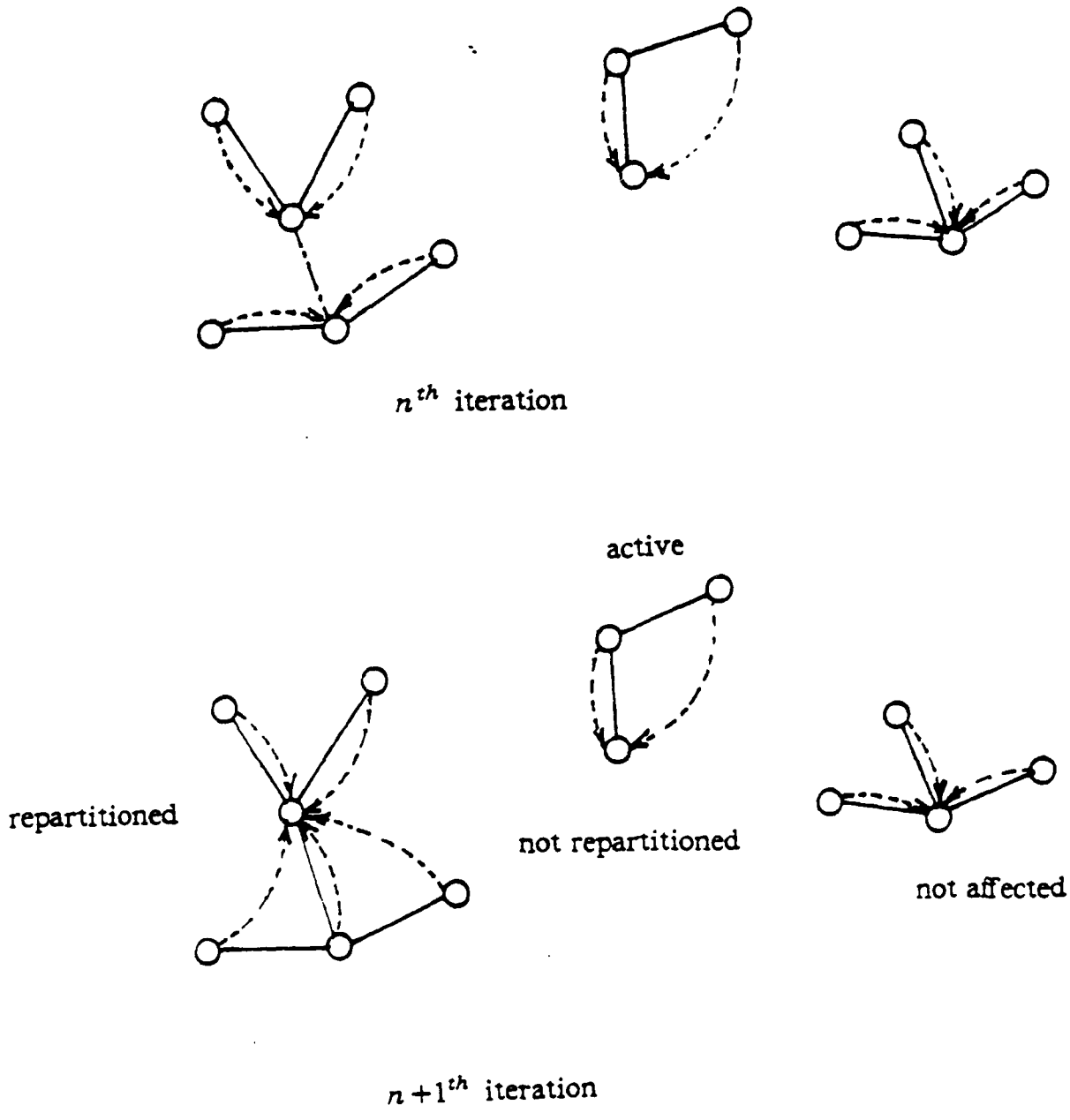


Fig. 20 Figure showing repartitioned, active and latent subcircuits

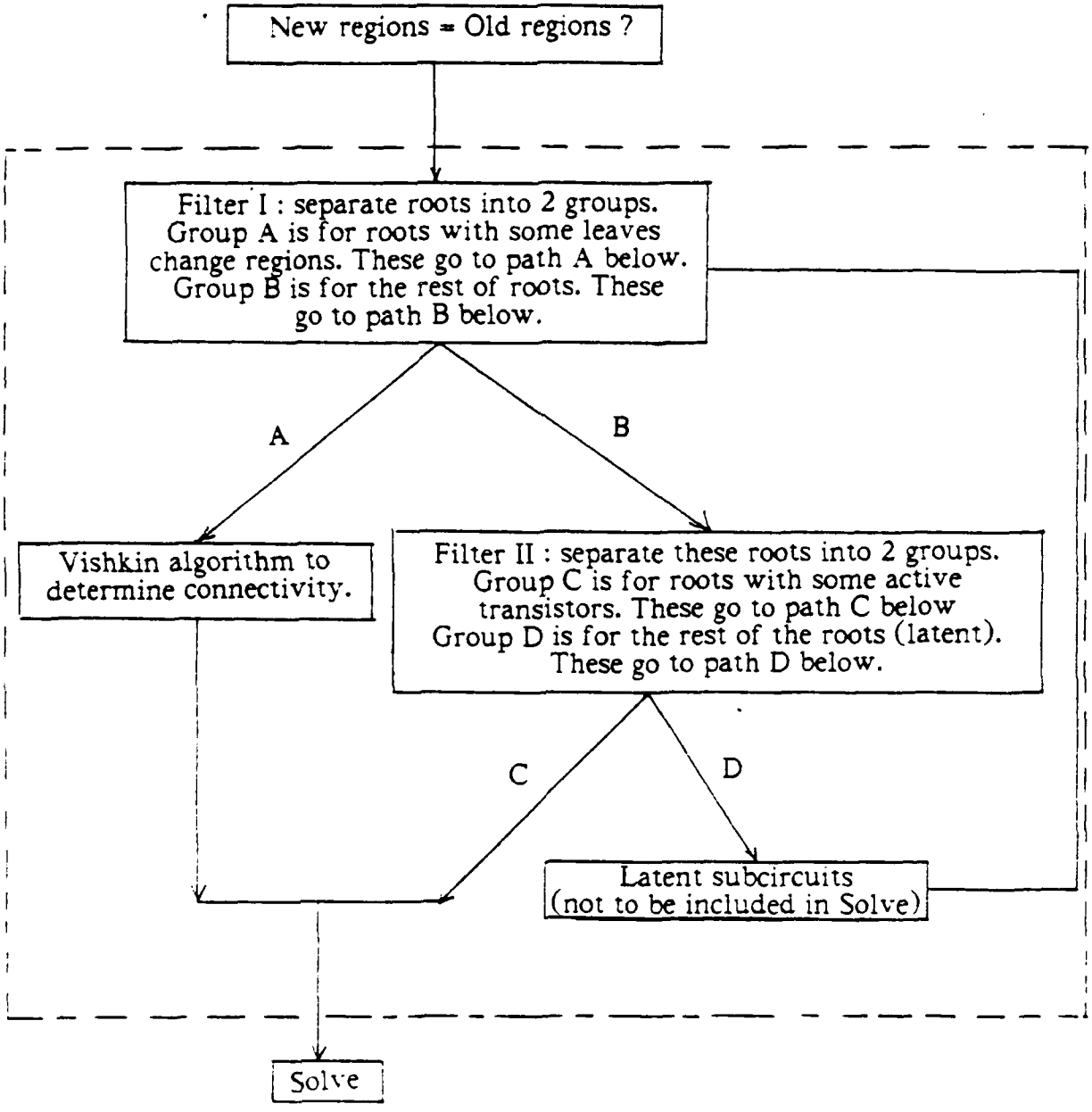


Fig. 21 Flowchart II : modification to Flowchart I

rest of the roots are latent subcircuits that are thrown to Filter I to be checked later if new edges formed affect these roots.

As mentioned before, unlike on the uniprocessor, on the parallel processors the partitioning is applied to the entire circuit. Most of the time the number of nodes in one connected component is less than three. For these small-size connected components a direct method is used to solve for the unknown variables. For larger connected components (number of nodes larger than three) the blocks are made as lower triangular as possible by applying Tarjan's depth-first search method [24] (to obtain strongly connected components within the large blocks) followed by the analysis sequencing method described in Chapter 2. These nearly lower triangular blocks are then solved. An example of the resulting matrix is shown in Figure 22. As mentioned earlier the large blocks do not occur often in circuits that we simulated.

In summary, the circuit is decomposed into blocks where each block is solved using the direct method by one processor. If the size of the block is small no reordering is done. If the size is large then the block is made as lower triangular as possible and then solved using one processor.

An alternative approach is to solve one block using all the available processors; that is, the unknown variables in one block are determined in parallel. Parallel numerical linear algebra such as described in [50] is needed. The drawback of this method is that in many cases the sizes of the blocks are small. This means there are more processors assigned to a block than needed to solve for the unknowns in that block. Therefore, there would be idle processors most of the time.

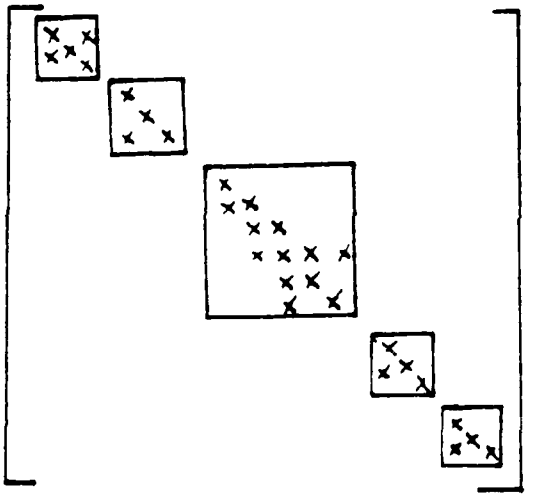
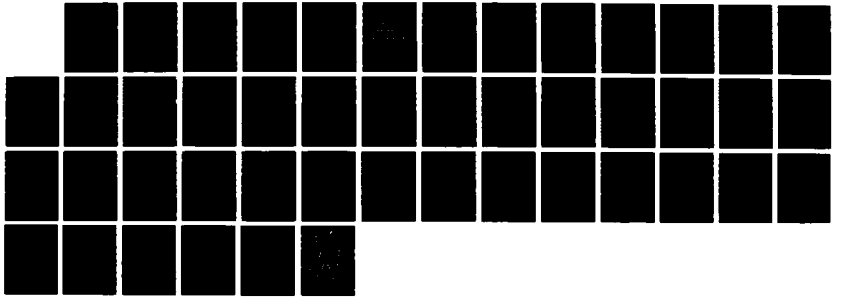
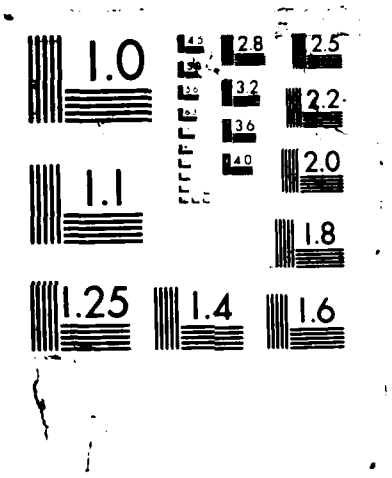


Fig. 22 Typical matrix after dynamic partitioning

If floating capacitors exist in the circuit then in the matrix the floating capacitors will create feedback and feedforward terms within and between the diagonal blocks created by the partitioning. Unlike in the uniprocessor case where the Gauss-Seidel relaxation method is employed, in the parallel case the Gauss-Jacobi relaxation method is applied.

AD-A191 328 PIECEWISE LINEAR APPROACH FOR TIMING SIMULATION OF VLSI 2/2
(VERY-LARGE-SCALE..(U) ILLINOIS UNIV AT URBANA
COORDINATED SCIENCE LAB O TEJAYADI DEC 87
UNCLASSIFIED UILU-ENG-87-2279 N00014-84-C-0149 F/G 20/6 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

CHAPTER 5

IMPLEMENTATION AND RESULTS

All three algorithms described in the preceding chapters have been implemented in computer programs to run on VAX 11/780 and SUN workstations. The parallel implementation of the dynamic partitioning method is for Alliant FX/8, a parallel-vector computer with 8 processors. The programs are written in FORTRAN and each has over 7800 lines of code.

The input file containing MOS network descriptions is similar to the one for MOTIS-C, except in our case the MOS network description can be in the transistor level or the predefined subcircuit level. The predefined subcircuits are nand, nor, inverter, and-or-inverter, and pass transistor net.

For the uniprocessor implementation the following steps are performed in the preprocessing stage. For each type of devices a *pwl* table is generated automatically. If no device information is given then default values for typical long channel devices are used. Next, the circuit is partitioned into dc-connected subcircuits [15]. If there exist floating capacitors then each one is checked if it is larger than the sum of the grounded capacitors to which the floating capacitor is connected. If it is, a dc-path is assumed to exist between the two nodes for partitioning purposes. If the capacitors are *pwl*, worst case values are assumed. This worst-case partitioning is performed only once and is based on the worst-case graph condition of the transistors (Figure 13e). Each subcircuit

representing a predefined or a dc-connected component is replaced by a node in a graph representing the circuit. The strongly connected components of the graph are identified using Tarjan's depth-first search described in Chapter 4. Then, analysis sequencing is performed on the new acyclic graph where each scc has been replaced by a new node. The strongly connected components are solved using the dynamic partitioning method, while each subcircuit of the rest of the circuit is solved using the direct method. If the user knows that some simple subcircuits, such as nand, nor, inverter, and-or-inverter, are not a part of an scc, then in the input file the user can specify these simple subcircuits as gates. This causes the program to solve those simple gates using the fast *pwl* method described in Chapter 2. The dynamic partitioning method automatically partitions the scc into smaller, completely decoupled blocks. In the cases where the blocks are too large (size of block is larger than three) those blocks are made as lower triangular as possible by applying Tarjan's depth-first search approach and analysis sequencing method described in Chapter 4. In almost all cases in practice the dynamic partitioning breaks the feedback paths in the scc. Information of regions of transistors needed for dynamic partitioning is obtained during the equation formulation process when the conductances and current sources are fetched from the *pwl* device tables. Based on this knowledge of regions, the program determines the local connectivity of the transistors. This local connectivity in turn is used to determine the global connectivity of the transistors in the scc by applying depth-first search [14]. This

depth-first search is not costly, since the size of an scc is usually not large. The subblocks are now solved in a sequence which usually does not include any feedback; and thus convergence is obtained in one sweep.

Waveforms of some examples are shown. The first one is a 5-stage ring oscillator circuit (Figure 15) containing floating capacitors. The example is used to show that fairly accurate results are obtained by the *pwl* method. Waveform SPICE is obtained by SPICE using level 1 model with external capacitances between any two adjacent nodes included. Waveform PWLFULL is obtained by using the *pwl* approximation and solving the entire circuit without partitioning or relaxation. Waveform PWLRELAX is obtained by dynamic partitioning and relaxation iteration to take into account the effects of floating capacitances between subcircuits. From the figures one can conclude that the *pwl* method gives accurate waveforms.

The second example is a tally circuit (Figures 24-25). Worst-case partitioning would define the entire circuit as one block while dynamic partitioning decomposes the circuit into small subblocks that can be solved separately; as a result, computation time is reduced.

The third example is the 10-stage inverter circuit (Figure 26). The output of the first, fourth, seventh and tenth inverters, together with SPICE waveforms, are shown in Figure 27. In this example the circuit is specified as inverter gates and the fast *pwl* method is applied. Note that for simple gates such as an inverter the fast *pwl* method is fairly accurate.

The fourth example is a full-adder circuit containing pass transistors (Figure 28). The waveforms of the sum and carry nodes are shown in Figure 29.

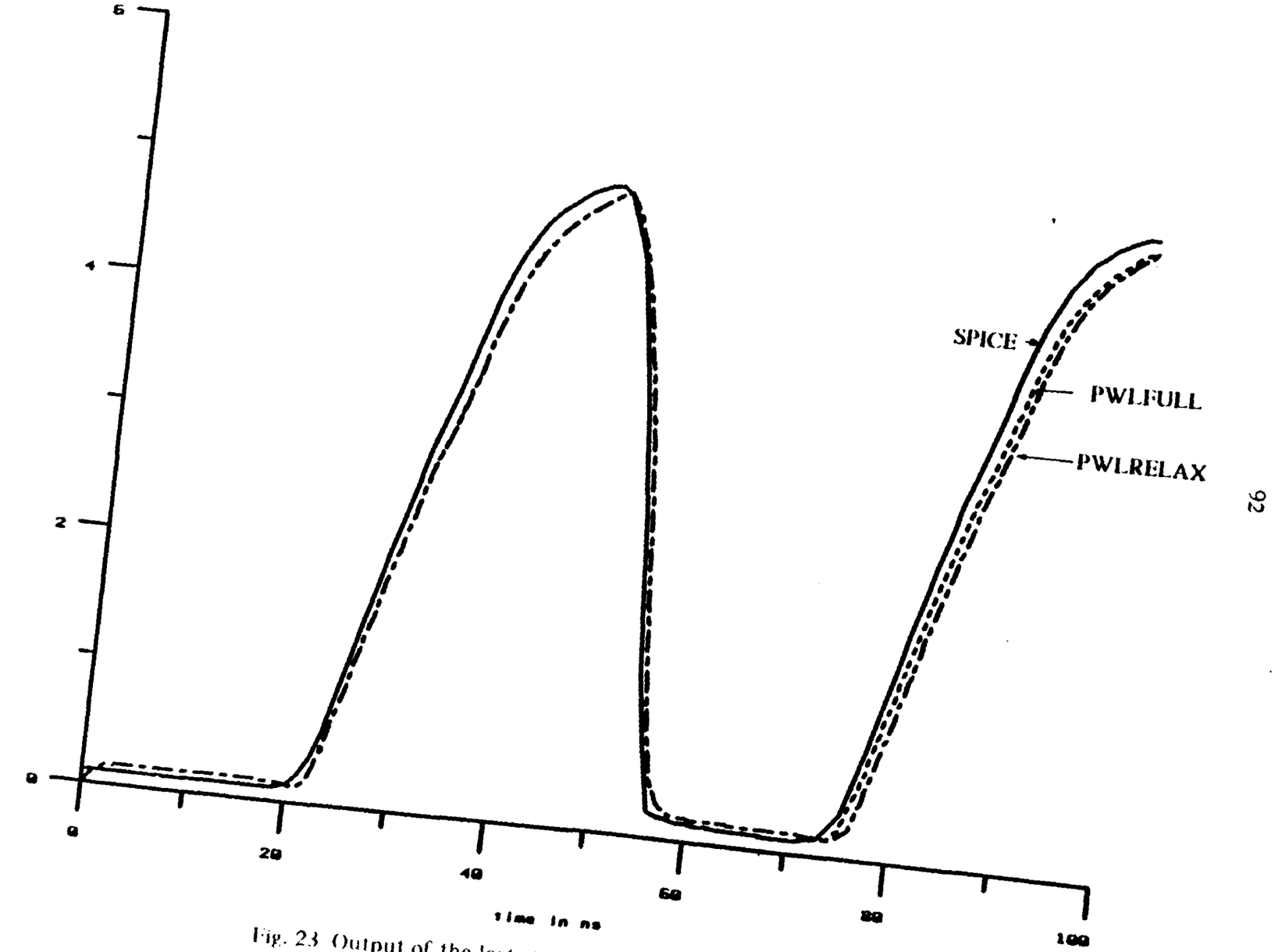


Fig. 23 Output of the last stage of 5-stage ring oscillator

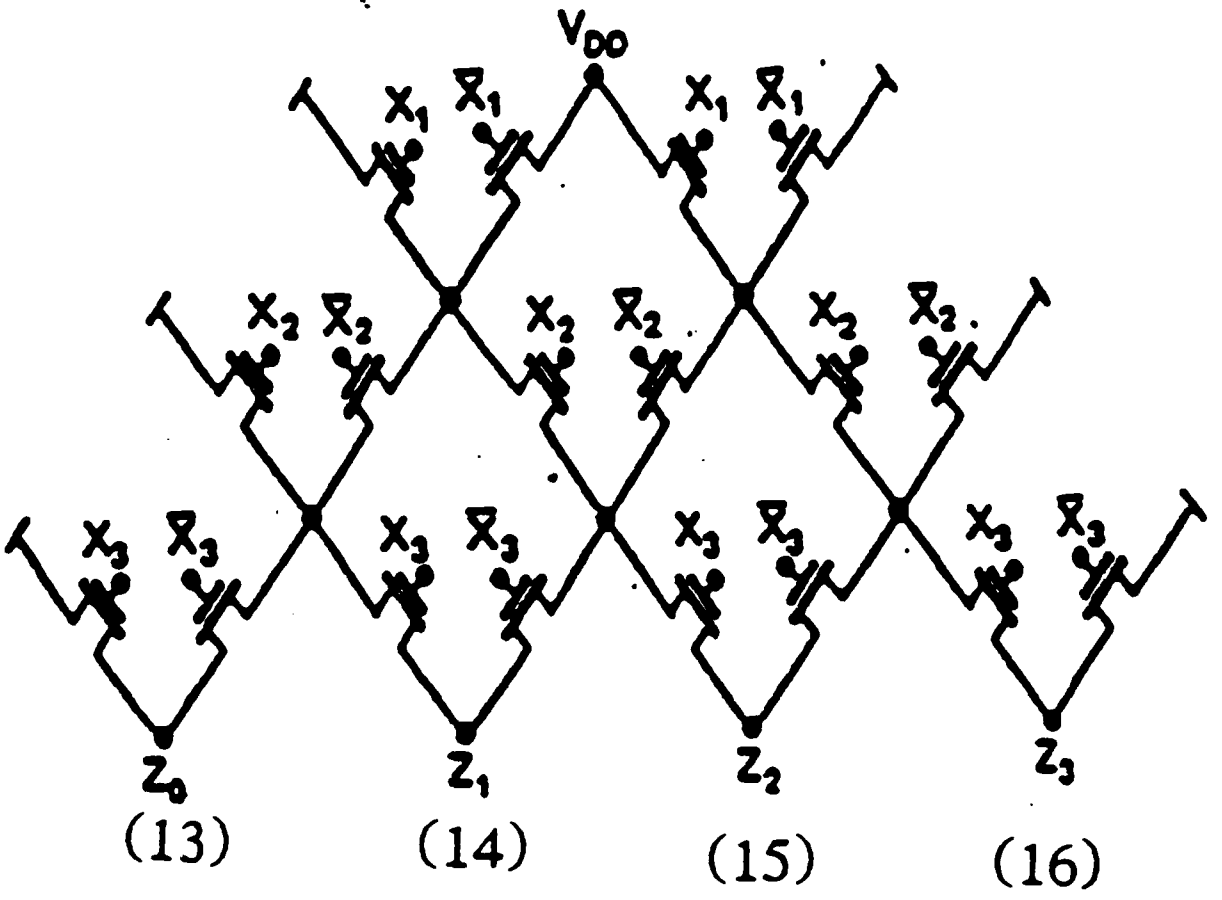


Fig. 24 Tally circuit

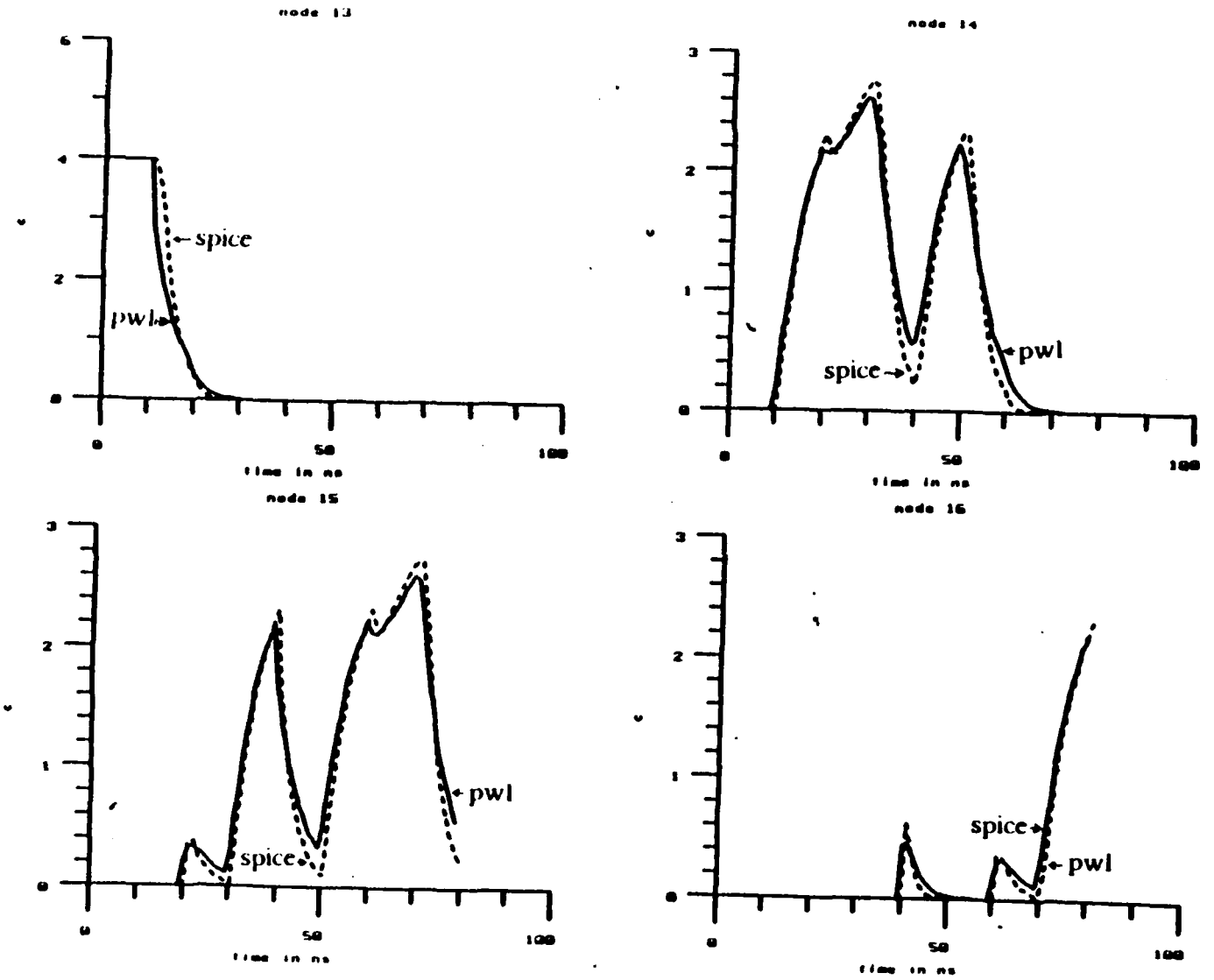


Fig. 25 Output of the tally circuit

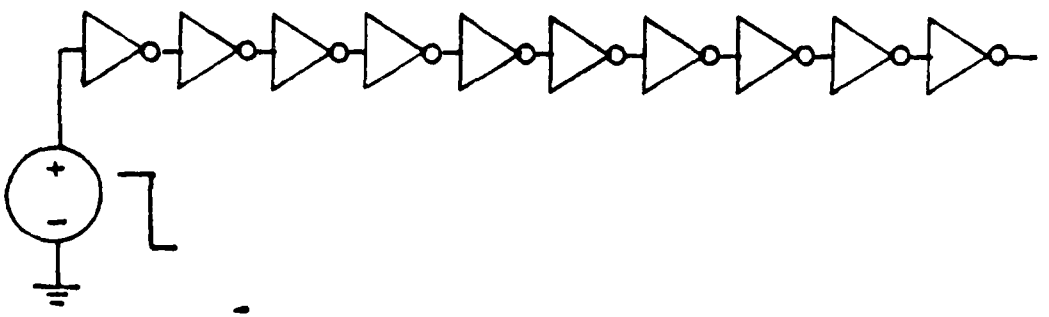


Fig. 26 10-stage inverter circuit

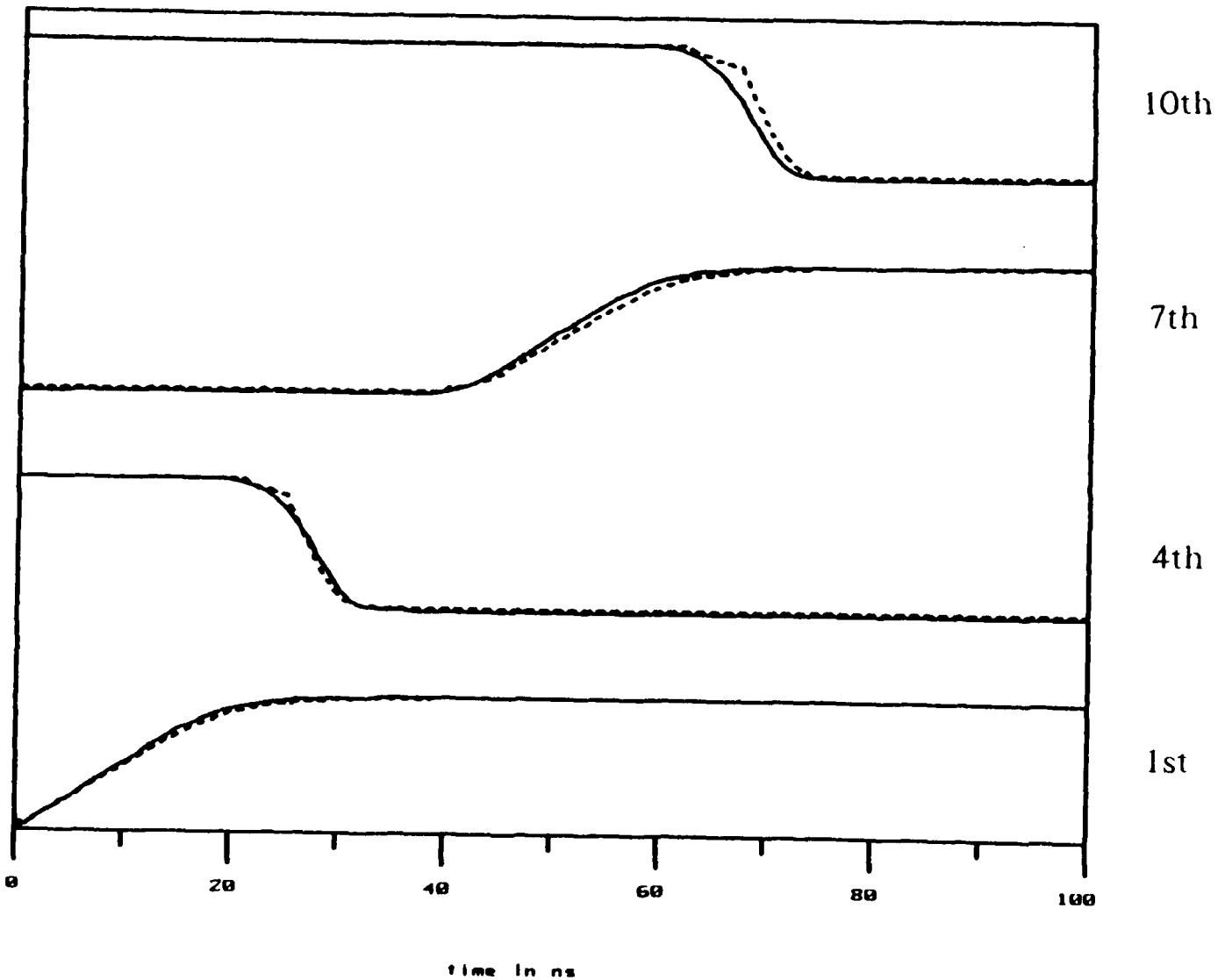


Fig. 27 Waveforms of the circuit in Fig. 26

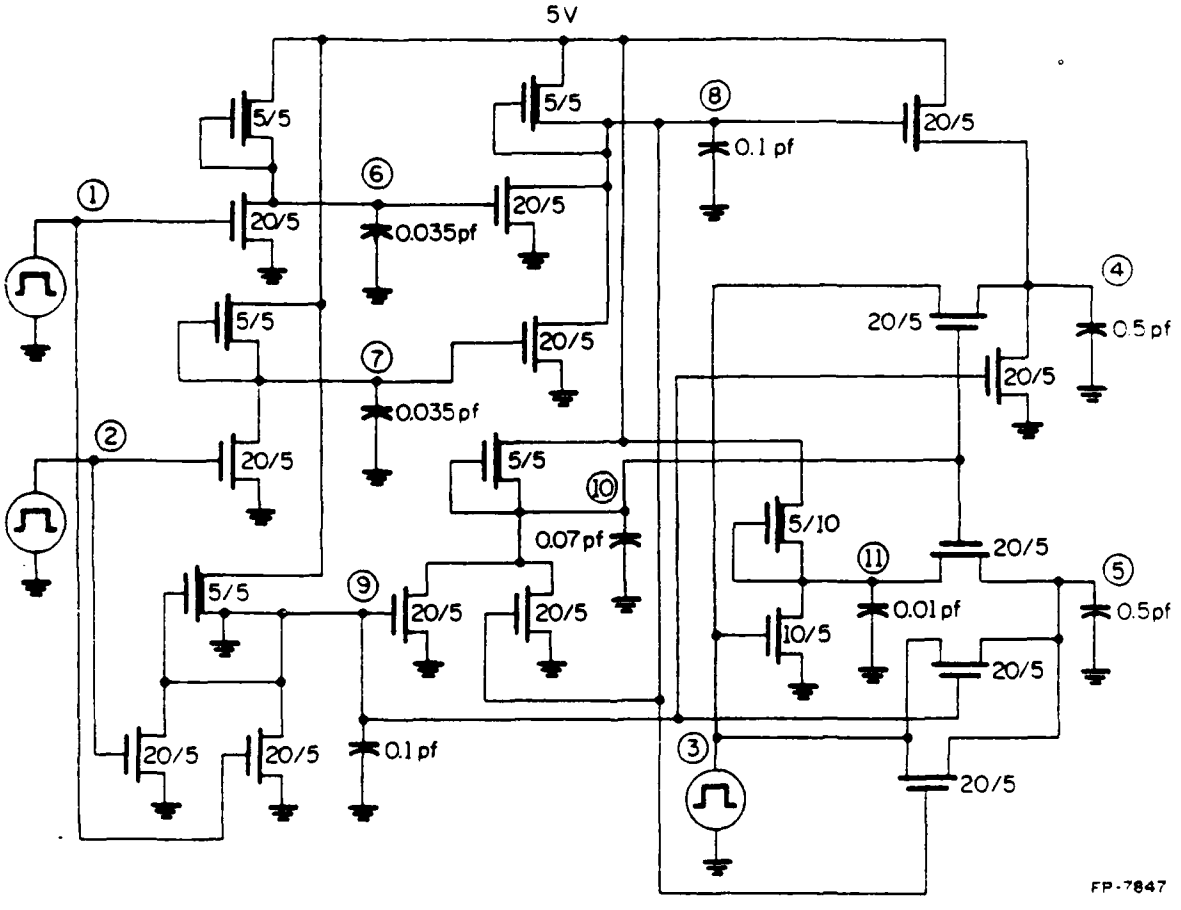


Fig. 28 Full adder with pass transistor

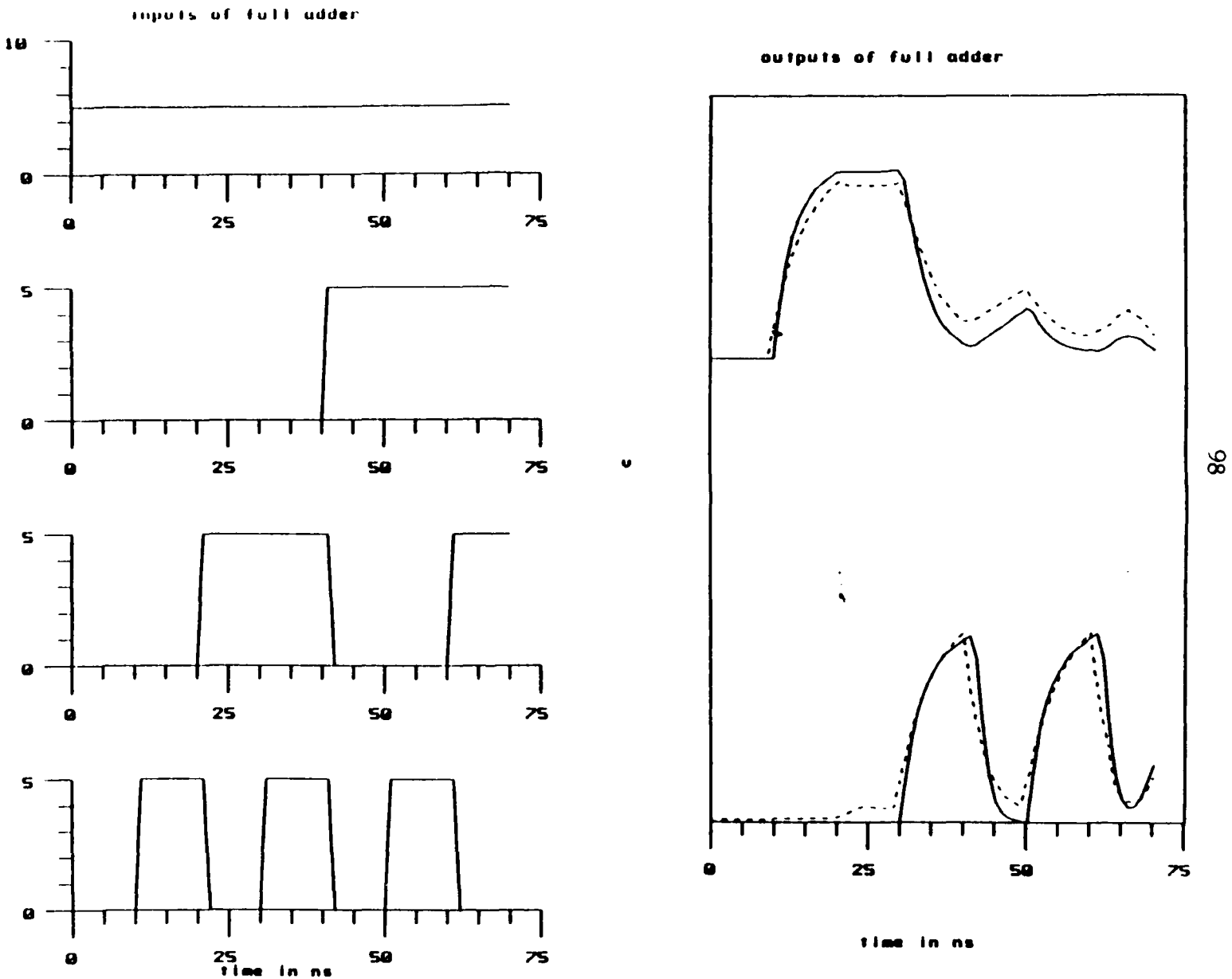


Fig. 29 Output of the II adder

The last example is the pla circuit (Figure 30). The waveforms of the output of the last inverters of the pla and SPICE waveforms are shown in Figure 31. This pla contains a strongly connected component which is solved using the dynamic partitioning method. The rest of the circuit, which consists of simple gates, is solved using the fast *pwl* method.

It can be seen from the figures that the *pwl* approximation is quite accurate compared to SPICE.

The computation time as compared to SPICE is shown in the following table.

Table III : Comparison of the *pwl* method and SPICE

		Analysis time		
circuit	devices	dynamic partitioning	no dynamic partitioning	SPICE
5-stage ring oscillator (no floating capacitor)	11	1.10s	1.417s	50.13s
5-stage ring oscillator (with floating capacitor)	11	1.17s	3.000s	45.20s
tally circuit	18	2.550s	3.167s	132s
pla	149	6.383s	14.22s	977s
cmos alu	142	4.171s	22.77s	n.c.

n.c. : no convergence

The table shows simulation results performed on some circuits. One observes that computation time is reduced when the dynamic partitioning is applied to

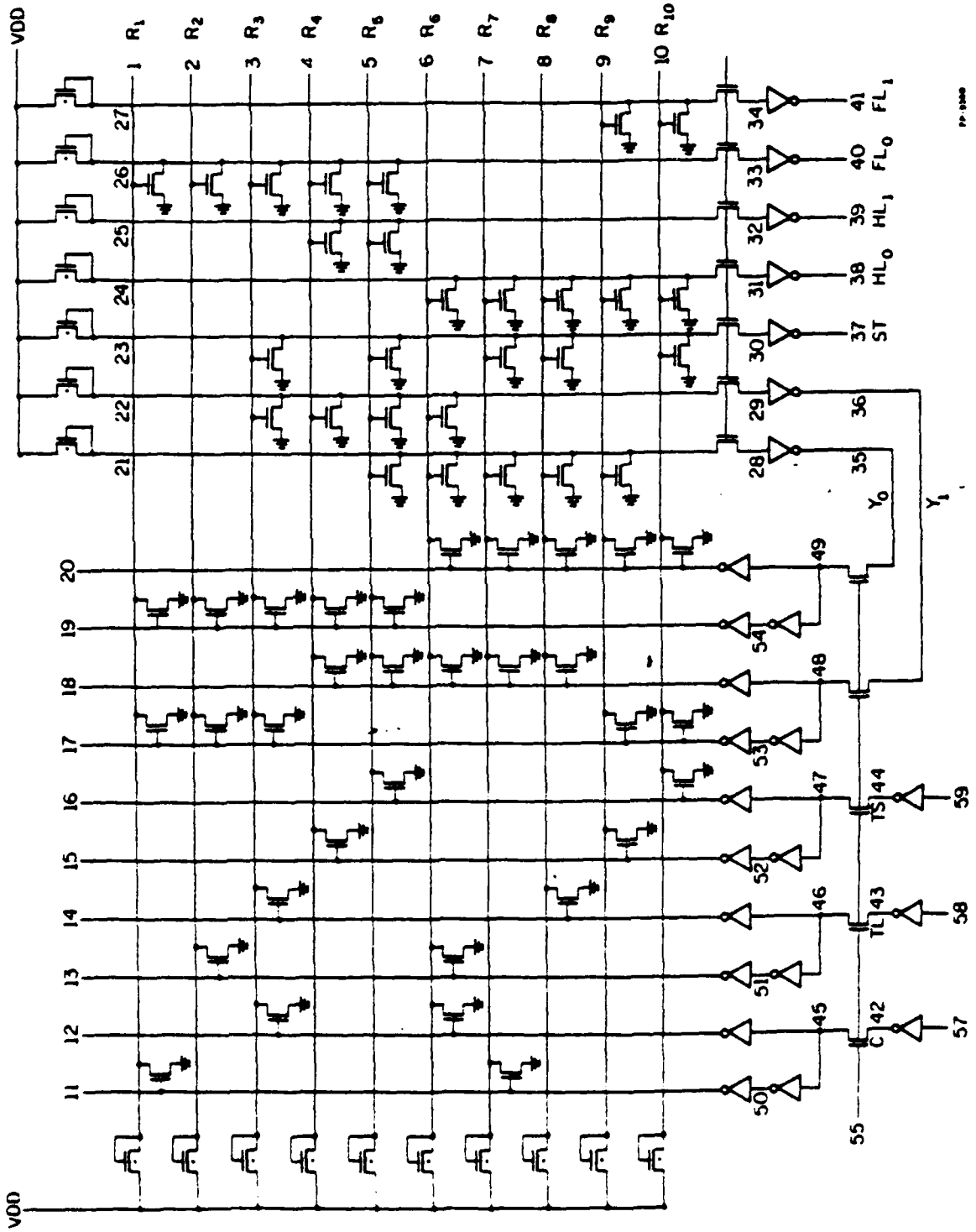


Fig. 30 Pla circuit

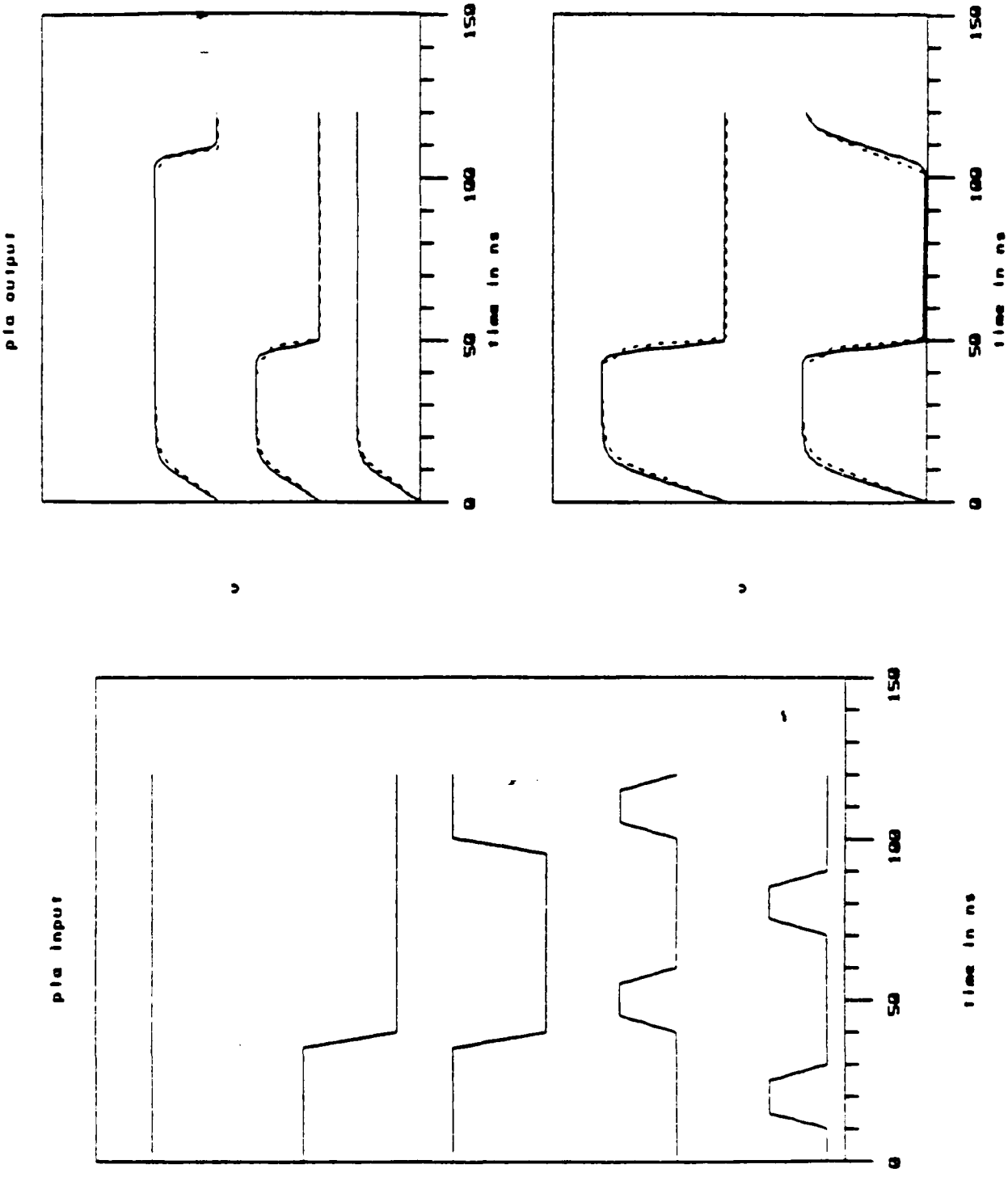


Fig. 31 Output of the pla

the circuits. For small circuits (less than 50 transistors) the speedup is about 40 as compared to SPICE. For a larger circuit, such as the pla, the speedup is over 100.

Computation time comparison with respect to RELAX2 [26], which applies worst-case partitioning method , is shown in Table IV. The table shows that *pwl* method is more than 10 times faster than RELAX2, even for these relatively small circuits.

Table IV : Comparison of the *pwl* method and RELAX3.2

Analysis time		
circuit {number of devices}	pwl with dynamic partitioning	RELAX3.2
5-stage ring oscillator (no floating capacitor) {11}	1.10s	14.02s
pla {149}	6.383s	155.36s

To obtain the rate of growth of computation time vs. the number of devices, an n-stage ring oscillator circuit is simulated. The CPU-times for the analysis times for various n are shown in Table V and plotted in Figure 32. We observe that the time grows fairly linearly as n increases.

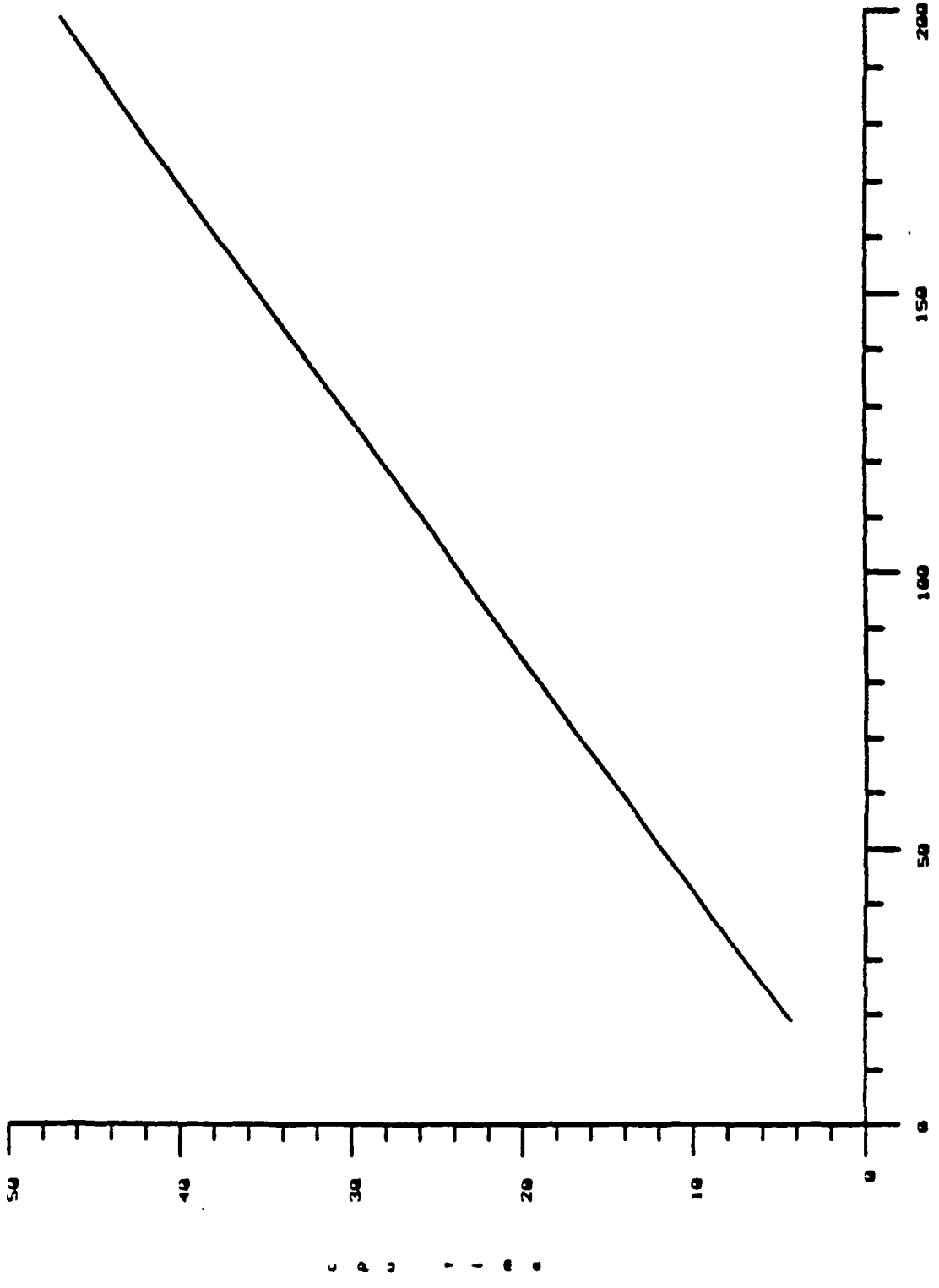


Fig. 32 Plot of computational complexity

Table V. Computational complexity

CPU-time vs. n	
Number of devices (n)	Analysis time in s
19	4.367
39	9.183
59	13.933
79	18.567
99	23.183
119	27.717
139	32.55
159	37.167
179	41.833
199	46.383

From the table above one can conclude that the CPU time taken by *pwl* method (method 3) grows linearly with the circuit size.

The dynamic partitioning method (method 3) implementation on the Alliant FX/8 is similar to the one for the sequential computer. The difference is that in parallel implementation there is no need to partition the circuit into dc-connected components at the outset. Instead, the partitioning is applied to the entire circuit. Also, there is no need to sequence the partitioned subcircuits since they are either completely decoupled from one another, or a Gauss-Jacobi method is used when a small capacitive coupling exits between subcircuits. However, the parallel implementation contains a locking mechanism that the sequential one does not have.

The following table shows the results of two circuits run on the Alliant FX/8. The circuits chosen consist of more than 100 transistors and are expected

to show some simulation speed advantage on the parallel computer due to the presence of large, strongly-connected components in the circuits. Speedup obtained for the pla circuit is over 600 times as compared to SPICE, while the speedup of the barrel-shifter circuit is over 400 times as compared to SPICE. The SPICE results are from a uniprocessor implementation. Compared to 1 processor, using 8 processors is over 3 times faster (for pla circuit), about 5.7 times faster (for the barrel shifter) and about 5.3 times faster (for the digital filter); this means, for the pla, the efficiency of processor utilization is over 37 percent, for the barrel shifter the efficiency is about 71 percent and for the digital filter it is 66 percent.

Table VI : Analysis time on the parallel processors

Analysis time on Alliant FX/8				
circuit	devices	dynamic partitioning (8 processors)	dynamic partitioning (1 processor)	SPICE
pla	149	1.14s	5.433s	977s
barrel shifter	256	1.983s	11.3s	862s
digital filter	698	11.6s	61.31s	-

Another version of the program contains filtering routines to do selective repartitioning and latency checks. The aim is to do repartitioning and solve only on some part of the circuit. For the pla circuit the computation time is reduced from 1.417 seconds to 1.1 seconds. For the barrel shifter no speedup is obtained; this is due to a large number of repartitioning and solving. The time spent on selective repartitioning and latency checks for the pla is 0.133 seconds (or about 12 percent of the total CPU time), while for the barrel shifter it is

0.28 seconds (or about 14 percent of the total CPU time). For the digital filter the time spent on the selective repartitioning and latency checks is 2.717 seconds (or approximately 20 percent of the total CPU time).

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

The piecewise linear approach, as described in this thesis, is an attractive method for solving circuit problems. This is due to the fact that simplified (*pwl*) transistor models are used yielding a lower memory requirement and faster computation time, and yet the method produces results that are close to those from other circuit simulators. Moreover, convergence is guaranteed in the *pwl* Katzenelson method and its variants.

A *pwl* MOS transistor model approximation is described in Chapter 2 and Appendix A. The model contains two parts: namely, the gate-drain and the gate-source parts. Each part consists of a current source, a resistor and a dependent current source. The dependent current source is inserted to satisfy the requirement that the sum of currents at the gate node is equal to zero. Higher order effects are modeled by the use of tabulated nested functions, as described in Appendix A. The *pwl* approximation method is also easily applicable to the Ebers-Molls bipolar transistor model.

Two *pwl* methods are also described in Chapter 2. The first is a modification of the *pwl* method on simplices. The method is suitable for circuits that demand more accurate analysis. The idea is similar to the *pwl* method developed by Katzenelson, except in this method, rather than finding boundary crossings, a vertex to be removed is selected. This vertex removal process is

much simpler than determining which boundary is crossed. The original method is general and slow. For timing analysis some speedup is obtained by piecewise linearizing the transistor model. Even after piecewise linearization of the model and parallelization of the calculation of the matrix entries, the method is not fast enough compared to the standard circuit simulator SPICE. The second method is a fast *pwl* method where the solution is obtained by partitioning the circuit into one-way subcircuits, where each subcircuit has one output with one capacitor lumped at the output. The waveform solution at the output of each subcircuit is found by using *pwl* Thevenin's equivalent circuit. Waveform relaxation is applied when feedback exists among the subcircuits. The method is fast and fairly accurate for simple circuits such as nand, nor and inverters. Larger circuits such as pass transistor networks require direct methods, since the fast *pwl* method is not accurate enough and tends to become slow.

Described in the third chapter is a new idea of dynamically partitioning the *pwl* circuit. The method is fast because the partitioning is based on comparing integers representing the gate-source and gate-drain regions of a transistor. Simulation results on a typical example show that more than two orders of magnitude speedup is obtained. The dynamic partitioning method is suitable for solving strongly connected components, or dc-connected subcircuits where the number of transistors is large. Smaller subcircuits can also be solved this way, or with the fast *pwl* method described in Chapter 2, or with the direct method. Another advantage of the dynamic partitioning method is its suitability for parallel implementation. This is described in Chapter 4. The local connectivity,

global connectivity and solving the dynamically partitioned subcircuits can be performed in the concurrent mode.

The detailed parallel implementation of the dynamic partitioning method is described in Chapter 4. The simulation of a number of circuit examples shows good speedup and efficiency of utilizing the parallel processors. Because the dynamic partitioning method partitions the circuit into completely decoupled small subcircuits, the gain in computation speed is fairly linear as the number of available processors increases.

The implementation issues of the program PLATINUM which is based on the dynamic partitioning method is described in Chapter 5. Several waveform examples and comparison with respect to other simulators are given to show the validity of the method.

PLATINUM as an experimental timing analysis shows good results for MOS circuits. More enhancements to the program are needed; in particular, it could be extended to handle bipolar circuits. It should not be a difficult task, since the bipolar transistor model is already in the Ebers-Molls configuration which is exactly what is needed for applying the dynamic partitioning method. Future work would also involve more testing on larger circuits and on other types of technology such as gallium arsenide circuits. Also, since the method is highly parallelizable, it would be interesting to implement the method on a massively parallel machine, similar to the recent parallel implementation of the relaxation method [48].

An interesting future work is to incorporate the dynamic partitioning method described in this thesis into a simulator such as RELAX2 which

employs accurate transistor models. Although PLATINUM uses simple *pwl* transistor models, iterations are necessary when floating capacitors, such as from the gate-drain and the gate-source capacitances, exist in the circuit. Since iterations are performed even for the simple models, it would be a good idea to use more accurate transistor models.

The questions that need to be answered in this case are

1. How does the selection of breakpoints affect the number of iterations to reach convergence.
2. If the selection of breakpoints affects the iterations, would it help, for accuracy, to have multiple *pwl* models for each transistor. The multiple *pwl* model is based on a nested tabulated functional represented and is described in Appendix A. A tradeoff between speed and accuracy is an issue here.
3. Compared to the heuristic partitioning method currently used in RELAX2, how much speedup does the dynamic partition provide.

It is possible that when more accuracy is desired the dynamic partitioning method based on a simple pwl model can be used to partition the circuit, while more accurate functional models are used in formulating and solving the equations.

APPENDIX A

SHORT CHANNEL *PWL* TRANSISTOR MODEL

This Appendix contains the more complete Meyer's model that includes the effect of the body-source and body-drain voltages. The simplified Meyer's model is given in Chapter 2. Instead of using

$$f(V_{GX}) = (V_{GX} - V_T)^2$$

one needs to use [25]:

$$f(V_{GX}) = (V_{GX} - V_T)^2 + \frac{4}{3} k (V_{XB} + 2^* \Phi_f)^{3/2} \qquad \text{(A.1)}$$

$$k = t_{ox} / \epsilon_{ox} (2q\, \epsilon_{Si} N)^{1/2}$$

where V_{XB} is the x to body voltage (x is either source or drain), Φ_f is the Fermi potential of the substrate, ϵ_{Si} is the permittivity of the substrate, N is the substrate concentration, and k is a constant. Note that the inclusion of the body effect preserves the one-dimensionality of the tables. Only one additional table for the body effect is needed to represent the second part of (A.1) for each device. The short channel effects on the threshold voltage V_T and the mobility μ_{EFF} can also be easily included in the tabular representation. The threshold voltage V_T is given by [55]:

$$V_T = V_{FB} + 2\Phi_f - \sigma V_{DS} + \gamma F_S (2\Phi_f - V_{BS})^{1/2} + F_N (2\Phi_f - V_{BS}) \qquad \text{(A.2)}$$

where

V_{FB} = *flatband voltage*

F_S = *correction factor for short channel effects*

F_N = *correction factor for narrow channel effects*

γ = *bulk threshold parameter*

σ = *coefficient of static feedback*

$$= ETA\frac{\Omega}{C_{ox}L^3}, \text{ where}$$

ETA = *constant static feedback effect parameter*

Ω = *empirical constant*

C_{ox} = *oxide capacitance*

L = *channel length*

The mobility μ_{EFF} is defined in two regions as follows:

For the saturation region :

$$\mu_{EFF} = \mu_S = \frac{UO}{1 + THETA(V_{GS} - V_T)} \tag{A.3}$$

while for linear region :

$$\mu_{EFF} = \frac{\mu_S}{1 + \frac{\mu_S}{VMAX*L}V_{DS}} \tag{A.4}$$

UO is the surface mobility, THETA is the empirical mobility modulation parameter and VMAX is the maximum drift velocity of the carriers. After taking into account the short channel effects the equation becomes as follows :

$$I_{DS} = K(V_{GS}, V_{DS})[\{V_{GS} - V_T(V_{BS}, V_{DS})\}^2 + \{\frac{4}{3}k(V_{SB} + 2*\Phi_s)^{3/2}\}$$

$$- \{V_{GD} - V_T(V_{BS}, V_{DS})\}^2 - \{\frac{4}{3}k(V_{DB} + 2*\Phi_s)^{3/2}\}] \tag{A.5}$$

$$\text{where } K(V_{GS}, V_{DS}) = \mu_{EFF}\epsilon_{ox}W/2t_{ox}L$$

A table based on Equation (A.5) would be multidimensional. To alleviate this difficulty we choose a set of discrete values of V_{BS},V_{DS} and V_{GS}. Each combination of V_{BS},V_{DS} and V_{GS} is used to obtain the threshold voltage V_T and the mobility μ_{EFF}. Then for the pairs V_T and μ_{EFF} we generate a set of values of conductances and current sources as the piecewise linearized model. For the n-type device the values of V_{DS} used in the table are 0,1,2,3,4,5. Similarly, the values for V_{BS} are -5,-4,-3,-2,-1,0 and for V_{GS} 0,1,2,3,4,5 ($V_{GS} \leqslant 0$ indicates the device is off). If more accurate results are desirable then more combinations of V_{BS},V_{DS} and V_{GS} are used to construct the table. The advantage of this method is that simple table lookup methods can be used to incorporate some of the short channel effects.

The above method can be considered as a nested modeling of the device. First, one determines the values of the variables at the lowest or deepest level of the equation. In our case the variables are V_T and μ. Then using these values, values of the variables of the higher level such as currents and conductances of the device model are calculated. Since in our case the independent variables V_{BS},V_{DS} and V_{GS} for the tables are determined a priori and *pwl* transistor approximations are tabulated in the preprocessing step, no calculation of a transistor characteristic is performed during the transient analysis, and hence the computation time is reduced. The calculation for the device elements is done during the preprocessing step and the values are stored in a table. This method of nested device modeling is similar to the one in [56]. The difference is as follows. In [56] the currents and conductances at various combinations of voltages are tabulated. Extrapolation and interpolation are necessary for any

combination of voltages outside the tabulated ones. In the approach described here the tabulated currents and conductances are the results of piecewise linearizing the original function. As a result, the currents and conductances for all combinations of voltages are defined, and therefore neither extrapolation nor interpolation is necessary.

To study the accuracy of the *pwl* approximation method described above, a CMOS latch with short channel transistors (1 micron length) is analyzed (Figure 33). SPICE outputs using simple model (level 1) and semiempirical model (level 3) are shown in Figure 34. It is clear that there is a noticeable difference between the SPICE outputs when using level 1 and level 3. Figure 34 also shows the outputs using the *pwl* model. It can be seen that there is good agreement between the output of SPICE level 3 and the output using the *pwl* model.

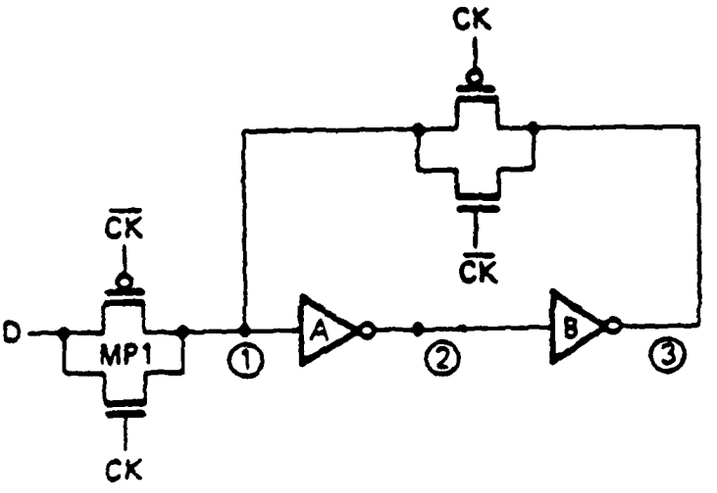


Fig. 33 Cmos d-latch circuit (1-micron channel)

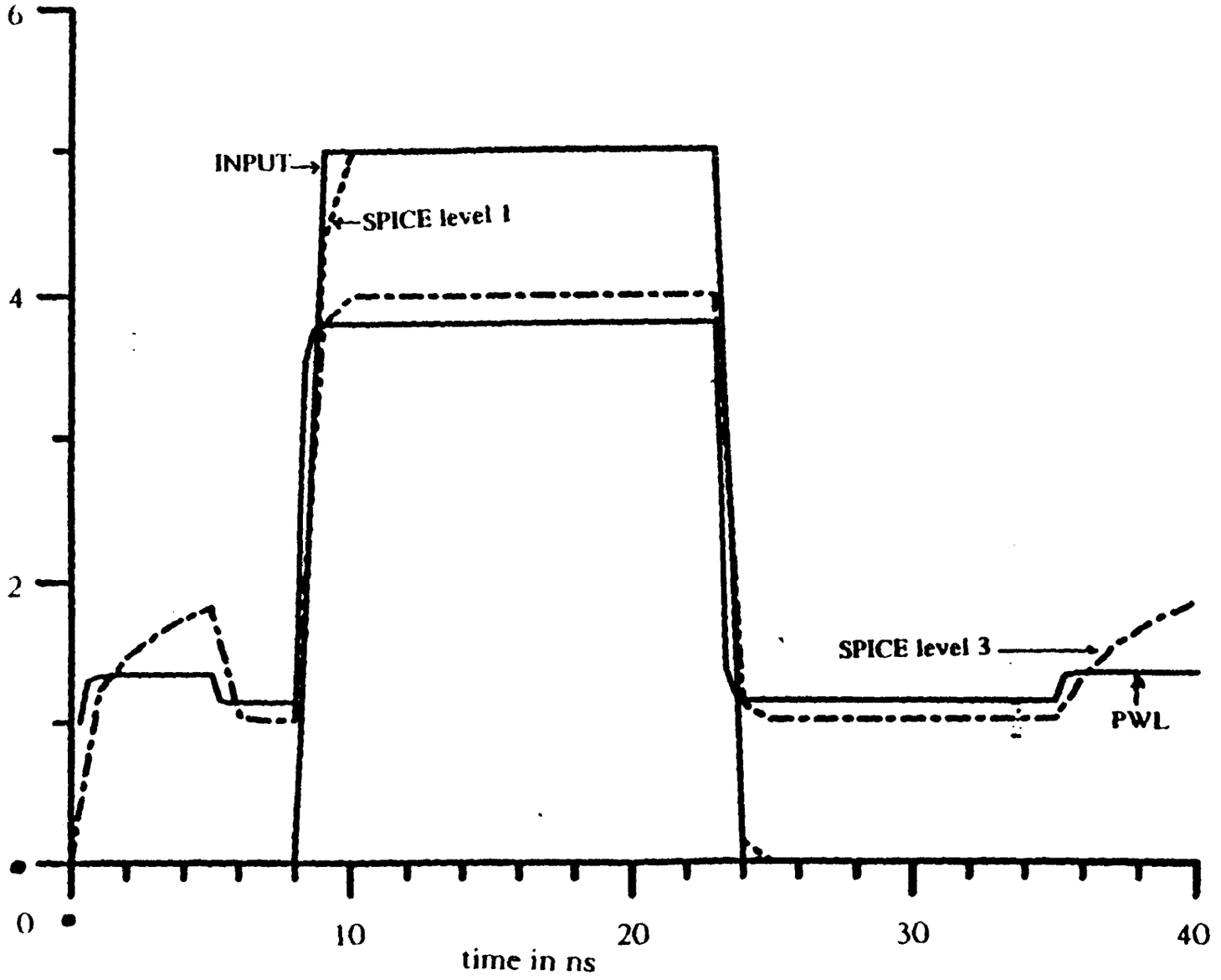


Fig. 34 Waveforms comparison of the circuit Fig. 3

APPENDIX B

DESCRIPTION OF THE PROGRAM PLATINUM

This appendix contains information on how to use PLATINUM: Piecewise LineAr TImiNg simUlation for Mos circuits. The input to the program, referred to as the circuit input file, is similar to the input file for the program PREMOS. PREMOS is a simulator developed by Wei [43]. PLATINUM is more general than PREMOS. Some of the features of PLATINUM are

1. It handles circuits described at the transistor or subcircuit level.
2. It has a built-in table for a typical *pwl* MOS driver, pull-up and pass transistor. The input file may contain user-specified transistor parameters which are used by the program to generate new *pwl* tables.
3. Capacitors are specified either from a node to ground or from a node to another node.

The types of subcircuits that can be handled by PLATINUM are nand, nor, and-or-inverter and pass transistor network. The model is described as

MODEL modnam type (parameters)

where modnam is a user-specified name, type is any one of the following : nand, nor, and-or-inverter, pass transistor, voltage source, and a set of appropriate parameters. The appropriate parameters for each type are (please refer to Figures B1-B4) :

TYPE PARAMETERS

nand wla wll ca cl cl

nor wlo wll co cl

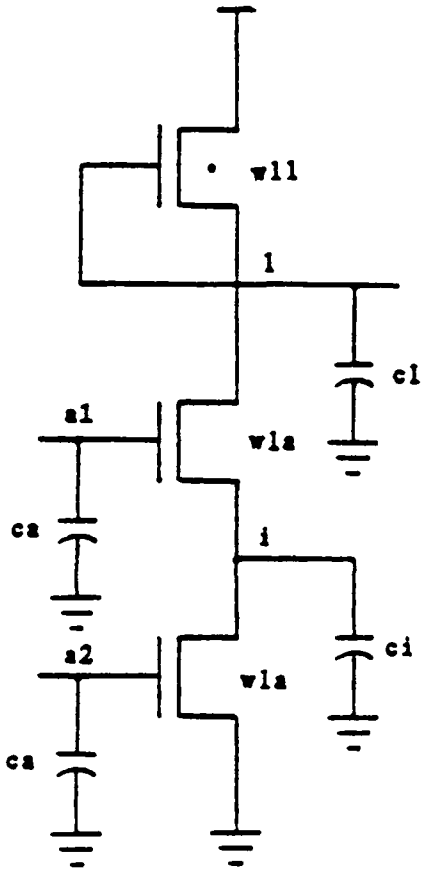


Fig. B1 Nand gate

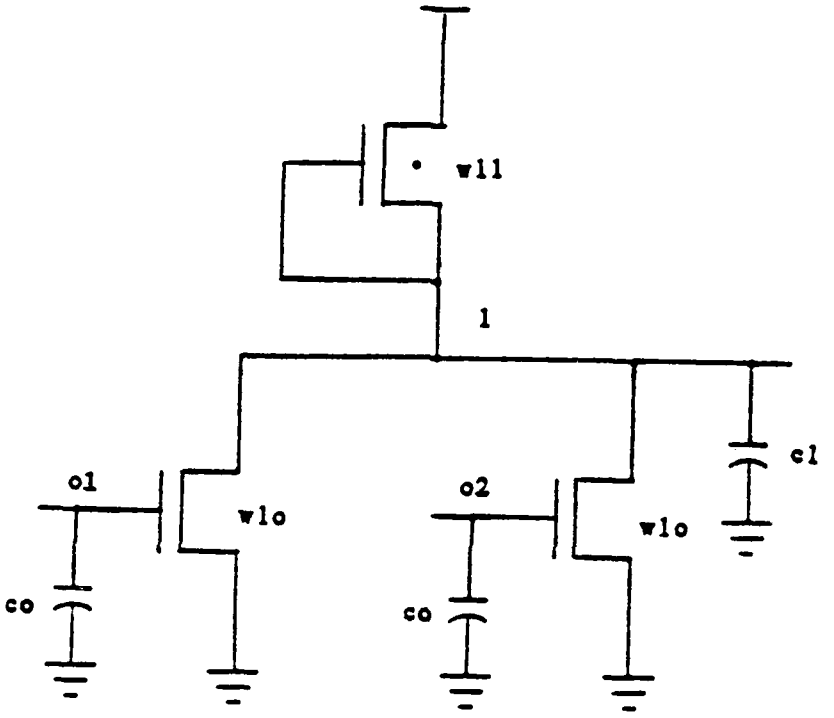


Fig. B2 Nor gate

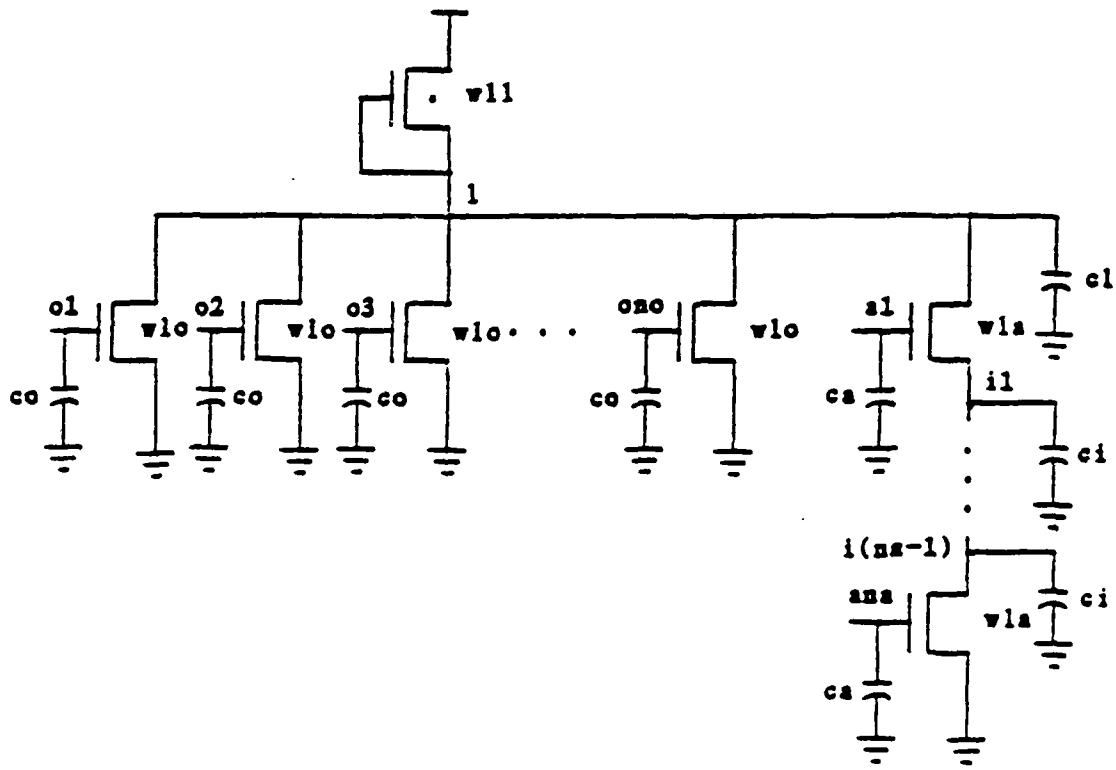


Fig. B3 And-or-inverter

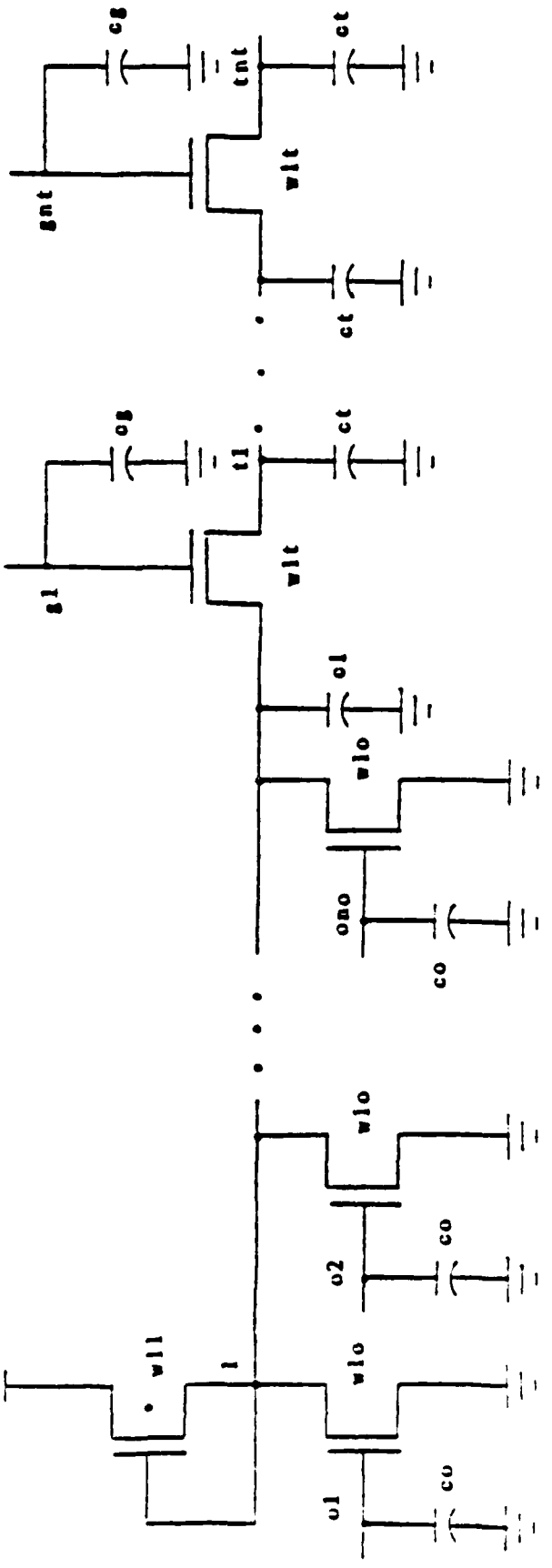


Fig. B4 N input nor gate with pass transistors

andor wla wlo wll ca co ci cl na no

trans wlo wll wlt co cl cg ct no nt

sourc vl v0 t0 tr t1 tf

For example, MODEL nd2 NAND (1 0.2 10f 10f 100f)

The models are used in the circuit description. The circuit description is of the form

name node1 node2 ... modnam

"name" is the name of the circuit element. The nodes "node1 node2 ... " contains the node connections. "modnam" is one of the model names. The node connections must follow the order given below :

TYPE ORDER OF NODE NUMBERS

nand a1 a2 i 1

nor o1 o2 1

andor a1 a2 ... o1 o2 ... 1 i1 i2 ... i(na-1)

caper n1 n2

sourc n-

Besides in the subcircuit level, a circuit can be described in the transistor level. The format for the transistor level description is

name drain gate source body trantype

where "name" is the name of the transistor element, "drain, gate, source body" are the MOS nodes, and "trantype" is the transistor type (such as PASS, DRIVER, LOAD). Drain and source nodes are interchangeable.

Besides circuit description the input file also contains options commands. The available options are

time tstop tstep

time is the command, tstop is the length of analysis time, and tstep is the time step.

preset (n1,v1) (n2,v2) ...

preset is the command to preset at the beginning of simulation a node to a specific voltage. n1,n2.... are the node numbers, and v1,v2.... are the node voltages.

send n1 n2 ...

send is the command to print out the node voltages. n1 n2 ... are the node numbers.

table (w,l,kappa,vt,v1,v2,v3,n)

table is the command to generate new table with user-specified parameters. w is the width, l is the length, kappa is the transconductance parameter, vt is the threshold voltage, v1,v2,v3 are the selected voltage breakpoints, and n is the type of transistor (n=1 is for driver, n=2 is for load and n=3 is for pass transistor).

end

end is a command indicating the end of the input file.

An example of a complete input file is given next. It is a pla circuit that is referred to in the thesis.

```
PLA finite-state machine implementing the light controller
*subcircuit model card
model inv nor2 (5 1 10f 100f
model nor3 andor(5 5 1 10f 10f 10f 100f 0 3)
model nor4 andor(5 5 1 10f 10f 10f 100f 0 4)
model notr1 trans(5 1 2 10f 100f 10f 50f 1 1)
```

```
model notr2 trans(5 1 2 10f 100f 10f 50f 2 1)
model notr4 trans(5 1 2 10f 100f 10f 50f 4 1)
model notr5 trans(5 1 2 10f 100f 10f 50f 5 1)
model pass passt
model cap capcr(50f)
model clk1 source (4 1 10n 5n 10n 5n)
model clk2 source (5 0 5n 5n 5n 5n)
* AND plane
x1 11 17 19 1 nor3
x2 13 17 19 2 nor3
x3 12 14 17 19 3 nor4
x4 15 18 19 4 nor3
x5 16 18 19 5 nor3
x6 12 13 18 20 6 nor4
x7 11 18 20 7 nor3
x8 14 18 20 8 nor3
x9 15 17 20 9 nor3
x10 16 17 20 10 nor3
* OR plane
x11 5 6 7 8 9 21 notr5
x44 21 56 28 pass
x12 3 4 5 6 22 notr4
x45 22 56 29 pass
x13 3 5 7 8 10 23 notr5
x46 23 56 30 pass
x14 6 7 8 9 10 24 notr5
x47 24 56 31 pass
x15 4 5 25 notr2
x48 25 56 32 pass
x16 1 2 3 4 5 26 notr5
x49 26 56 33 pass
x17 9 10 27 notr2
x50 27 56 34 pass
* output registers
x18 28 35 notr1
x51 35 55 49 pass
x19 29 36 notr1
x52 36 55 48 pass
x20 30 30 37 inv
x21 31 31 38 inv
x22 32 32 39 inv
x23 33 33 40 inv
x24 34 34 41 inv
* capacitors of pass trans
x56 28 0 cap
x57 29 0 cap
x58 30 0 cap
x59 31 0 cap
```

```
x60 32 0 cap
x61 33 0 cap
x62 34 0 cap
x66 48 0 cap
x67 49 0 cap
* input buffers
x25 57 42 notr1
x53 42 55 45 pass
x26 58 43 notr1
x54 43 55 46 pass
x27 59 44 notr1
x55 44 55 47 pass
x63 45 0 cap
x64 46 0 cap
x65 47 0 cap
* input registers
x28 45 45 50 inv
x29 46 46 51 inv
x30 47 47 52 inv
x31 48 48 53 inv
x32 49 49 54 inv
x33 50 50 11 inv
x34 45 45 12 inv
x35 51 51 13 inv
x36 46 46 14 inv
x37 52 52 15 inv
x38 47 47 16 inv
x40 53 53 17 inv
x41 48 48 18 inv
x42 54 54 19 inv
x43 49 49 20 inv
*input sources
va1 55 0 clk1 0 1 0 0 0 1 0 0 0 1 0 0 0 1
va2 56 0 clk1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
va0 57 0 clk2 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1
vb0 58 0 clk2 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
vc0 59 0 clk2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
*analysis requests
preset (35.0) (36.0)
time 120n 1n
send 37 38 39 40 41
v- 5
end
```

REFERENCES

[1] L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," *Electronics Research Laboratory Report #ERL-M520*, University of California, Berkeley, May 1975.

[2] P. Yang, I. N. Hajj and T. N. Trick, "SLATE: A Circuit Simulation Program with Latency Exploitation and Node Tearing," *Proceedings of the IEEE International Conference on Circuits and Computers*, pp. 353-355, October 1980.

[3] R. E. Bryant, "An Algorithm for MOS Logic Simulation," *LAMBDA*, vol. 1, no. 3, pp. 46-53, 1980.

[4] I. N. Hajj and D. Saab, "Symbolic Logic Simulation of MOS Circuits," *Proceedings of the IEEE International Symposium on Circuits and Systems*, Newport Beach, California, pp. 246-249, May 1983.

[5] B. R. Chawla, H. K. Gummel and P. Kozak, "MOTIS - An MOS Timing Simulator," *IEEE Transaction on Circuits and Systems*, vol. CAS-22, pp. 901-910, December 1975.

[6] S. P. Fan, M. Y. Hsueh, A. R. Newton and D. O. Pederson, "MOTIS-C: A New Circuit Simulator for MOS LSI Circuits," *Proceedings of the International Symposium on Circuits and Systems*, Phoenix, Arizona, pp. 700-703, April 1977.

[7] C. F. Chen, C. Y. Lo, H. N. Nham, and P. Subramaniam, "The Second Generation MOTIS Mixed Mode Simulator," *Proceedings of the 21st Design Automation Conference*, Albuquerque, New Mexico, pp. 10-17, June 1984.

[8] Y. P. Wei, I. N. Hajj and T. N. Trick, "A Prediction-Relaxation Based Simulator for MOS Circuits," *Proceedings of the IEEE International Conference on Circuits and Computers*, New York, September 1982.

[9] C. J. Terman, "RSIM - A Logic-Level Timing Simulator," *Proceedings of the IEEE International Conference on Computer Design*, New York, pp. 437-440, November 1983.

[10] V. B. Rao, T. N. Trick and I. N. Hajj, "A Table-Driven Delay-Operator Approach to Timing Simulation of MOS VLSI Circuits," *Proceedings of the International Conference on Computer Design*, New York, pp. 445-448, November 1983.

[11] E. Lelarasmee, A. E. Ruehli and A. L. Sangiovanni-Vincentelli, "The Waveform Relaxation Method for the Time Domain Analysis of Large Scale Integrated Circuits," *IEEE Transaction on Computer-Aided Design*, vol. CAD-1, no. 3, pp. 131-145, July 1982.

[12] I. N. Hajj and S. Skelboe, "Time Domain Analysis of Nonlinear Systems with Finite Number of Continuous Derivatives," *IEEE Transactions on Circuits and Systems*, vol. CAS-26, May 1979.

[13] R. J. Kaye and A. Sangiovanni-Vincentelli, "Solution of Piecewise Linear Ordinary Differential Equations Using Waveform Relaxation and Laplace Transform," *IEEE Transactions on Circuits and Systems*, vol. CAS-30, no. 6, pp. 353-357, June 1983.

[14] J. K. Ousterhout, "Crystal: A Timing Analyzer for Nmos VLSI Circuits," *Computer Science Division, Report #UCB/CSD83/115*, University of Cali-

fornia, Berkeley, January 1983.

[15] N. P. Jouppi, "Timing Analysis for Nmos VLSI," *Proceedings of the 20th Design Automation Conference*, pp. 411-418, June 1983.

[16] Y. H. Kim, J. E. Kleckner, R. A. Saleh, and A. R. Newton, "Electrical-Logic Simulation," *Digest 1984 International Conference on CAD*, pp. 7-9, November 1984.

[17] S. H. Hwang, Y. H. Kim, and A. R. Newton, "An Accurate Delay Modelling Technique for Switch-Level Timing Verification," *Proceedings of the 23rd Design Automation Conference*, pp. 227-233, June 1986.

[18] L. M. Vidigal, S. R. Nassif and S. W. Director, "CINNAMON: Coupled Integration and Nodal Analysis of MOs Networks," *Proceedings of the 23rd Design Automation Conference*, pp. 179-185, June 1986.

[19] V. B. Rao and T. N. Trick, "A New Approach to Processing Strongly Connected Circuit Blocks in a Waveform Relaxation Switch-Level Timing Simulator," *Proceedings of the IEEE International Conference on Computer Design*, October 1984.

[20] W. C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *Journal of Applied Physics*, vol. 19, pp. 55-64, January 1948.

[21] P. Penfield and J. Rubinstein, "Signal Delays in RC Tree Networks," *Proceedings of the 18th Design Automation Conference*, pp. 613-617, 1981.

[22] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, New Jersey: Prentice-Hall, Inc., 1975.

[23] T. Fujisawa and E. S. Kuh, "Piecewise Linear Theory of Nonlinear Networks," *SIAM Journal of Applied Math.*, vol. 22, no. 2, pp. 307-328, March 1972.

[24] R. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM Journal of Computing*, vol. 1, no. 2, pp. 146-160, June 1972.

[25] J. E. Meyer, "MOS Models and Circuit Simulation," *RCA Review*, vol. 32, pp. 42-63, March 1971.

[26] J. White and A .L. Sangiovanni-Vincentelli, "RELAX2: A Modified Waveform Relaxation Approach to the Simulation of MOS Digital Circuits," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 756-759, May 1983.

[27] C. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley Publishing Co., 1980.

[28] I. N. Hajj, K. Singhal, J. Vlach, "Efficient Analysis of Non-Linear Networks by Piecewise Linear Approximations," *Proceedings of the 11th Allerton Conference on Circuit Theory*, pp. 635-642, October, 1973.

[29] J. Katzenelson, "An Algorithm for Solving Nonlinear Resistive Networks," *Bell System Technical Journal*, vol. 44, pp. 1605-1620, October 1965.

[30] M. Ilic-Spong, I. N. Katz, J. Zaborszky, "Block Diagonal Dominance for Systems of Nonlinear Equations with Applications to Load Flow Calculations in Power Systems," *Mathematical Modeling*, vol.5, pp. 275-297, 1984.

[31] N. Deo, *Graph Theory with Application to Engineering and Computer Science*. New Jersey: Prentice Hall, 1974.

[32] L. O. Chua and P. M. Lin, *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques*. New Jersey: Prentice-Hall, Inc., 1975.

[33] J. White and A. Sangiovanni-Vincentelli, "Partitioning Algorithms and Parallel Implementations of Waveform Relaxation Algorithms for Circuit Simulation," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 221-224, June 1985.

[34] G. Marong and A. Sangiovanni-Vincentelli, "Waveform Relaxation and Dynamic Partitioning for the Transient Simulation of Large Scale Bipolar Circuits," *Proceedings of ICCAD*, pp. 32-34, November 1985.

[35] A. R. Newton, A. L. Sangiovanni-Vincentelli, "Relaxation-Based Electrical Simulation," *IEEE Transactions on Electron Devices*, pp. 1184-1207, vol. ED-30, no.9, September 1983.

[36] "Advanced Statistical Analysis Program (ASTAP)," IBM Corp. Data Proc. Div., White Plains, NY, Pub. no. SH20-1118-0.

[37] Q. J. Yu and O. Wing, "PLMAP - A Piecewise Linear MOS Circuit Analysis Program," *Integration, the VLSI Journal*, pp. 27-48, 1984.

[38] W. M. Penney and L. Lau, Eds., *MOS Integrated Circuits*. New York: Van Nostrand Reinhold, 1972.

[39] I. N. Hajj and K. K. Jung, "A Piecewise Linear Approach to Timing Analysis of VLSI Circuits," *International Symposium of Circuit and Systems*, 1984.

[40] M. J. Chien and E. S. Kuh, "Solving Nonlinear Resistive Networks Using Piecewise Linear Analysis and Simplicial Subdivisions," *IEEE Transaction on Circuits and Systems*, vol. CAS-24, no. 6, pp. 305-317, June 1977.

[41] R. Saleh and A .R. Newton, "Iterated Timing Analysis and SPLICE1," *Proceedings IEEE ICCAD Conference*, September 1983.

[42] N. Tanabe, H. Nakamura and K. Kawakita, "MOSTAP: An MOS Circuit Simulator for LSI Circuits," *Proceedings of IEEE International Symposium on Circuit and Systems*, pp. 1035-1038, April, 1980.

[43] Y. P. Wei, "Large-Scale Circuit Simulation," Ph.D. dissertation, University of Illinois, Urbana, Illinois, 1983.

[44] A. E. Ruehli, A. L. Sangiovanni-Vincentelli and N.B.Rabbat, "Time Analysis of Large-Scale Circuits Containing One-Way Macromodels," *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 766-770, April 1980.

[45] R. S. Varga, *Matrix Iterative Analysis*. New Jersey: Prentice-Hall, Inc., 1962.

[46] C. W. Ho, A. E. Ruehli and P. A. Brennan, "The Modified Nodal Approach to Network Analysis," *IEEE Transactions on Circuits and Systems*, pp. 504-509, vol. CAS-22, June 1975.

[47] M. P. Desai, "Block Time-Point Relaxation Algorithms for Circuit Simulation," *IEEE International Conference on Computer-Aided Design*, pp. 88-91, November 1986.

[48] D. M. Webber and A. Sangiovanni-Vincentelli, "Circuit Simulation on the Connection Machine," *Proceedings of the 24th Design Automation Conference*, Las Vegas, Nevada, pp. 108-113, August 1987. vol. 13, no. 4, pp. 524-547, October 1975.

[49] P. Yang and P. K. Chatterjee, "SPICE Modeling for Small Geometry MOSFET Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-1, no. 4, pp. 169-182, October 1982.

[50] D. Heller, "A Survey of Parallel Algorithms in Numerical Linear Algebra," *SIAM Review*, vol. 20, no. 4, pp. 740-777, October 1978.

[51] I. N. Hajj, "Sparsity Considerations in Network Solution by Tearing," *IEEE Transactions on Circuits and Systems*, vol. CAS-27, no. 5, pp. 357-366, May 1980.

[52] D. S. Hirschberg, A. K. Chandra and D. V. Sarwate, "Computing Connected Components on Parallel Computers *Communications of ACM*, pp. 461-464, vol. 21, no. 8, August 1978.

[53] C. M. Fiduccia and R. M. Mattheyses, "A Processor-Efficient Connected Components Algorithm," *Proceedings of 18th Annual Allerton Conference on Communication, Control and Computing*, October 1980.

[54] Y. Shiloac and U. Vishkin, "An O(log n) Parallel Connectivity Algorithm," *Journal of Algorithms* vol. 3, pp. 57-67, 1982.

[55] A. Vladimirescu and S. Liu, "The Simulation of MOS Integrated Circuits Using SPICE2," *Memorandum no. UCB/ERL M80/7*, February 1980.

VITA

Ongky Tejayadi was born in Bandung, Indonesia on October 16, 1959. He received his Bachelor of Science degree in Electrical Engineering from the University of Illinois at Urbana-Champaign in May 1981. He was awarded the Bronze Tablet when he graduated. In August 1981 he entered the graduate college of the University of Illinois. He received his Master of Science degree in Electrical Engineering in January 1983 and his Doctor of Philosophy degree in Electrical Engineering in January 1988.

He was a research assistant in the Semiconductor Group, working on ion implantation of indium phosphide under Professor B.G. Streetman from August 1981 until August 1982. From September 1982 until May 1983 he did research in photoluminescence of GaAs and AlGaAs and in fabrication of a double heterojunction bipolar transistor in the Molecular Beam Epitaxy Group. In June 1983 he joined the Digital and Analog Circuits Group to work on his doctoral thesis under Professor Hajj. In June 1987 he started working for Cray Research, Inc., in Chippewa Falls, Wisconsin. His research interests are in the areas of computer-aided design of VLSI circuits, and III-V compound semiconductor characterizations and fabrications.

END

DATE

FILMED

5-88

DTIC